32nd International Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2012, December 15–17, 2012, Hyderabad, India

Edited by

Deepak D'Souza

Telikepalli Kavitha

Jaikumar Radhakrishnan

Editors

Deepak D'Souza
Dept. of Computer Science & Automation
Indian Institute of Science
Bangalore 560012
India
deepakd@csa.iisc.ernet.in

Jaikumar Radhakrishnan and Telikepalli Kavitha
School of Technology and Computer Science
Tata Institute of Fundamental Research
Homi Bhabha Road, Mumbai 400005
India
jaikumar@tifr.res.in, kavitha@tcs.tifr.res.in

ACM Classification 1998
D.2.4 Software/Program Verification, F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, F.4.3 Formal Languages

ISBN 978-3-939897-47-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-43-9.

Publication date
December, 2012

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at .

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2012.i

ISBN 978-3-939897-47-7 **ISSN 1868-8969** **www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

www.dagstuhl.de/lipics

Contents

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, T. Kavitha, and J. Radhakrishnan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Session 2A

Session 2B

Session 3A

Session 3B

Session 4A

Session 4B

Session 5A

Session 5B

Session 6A

Session 6B

Session 7A

Session 7B

Preface

The 32nd international conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012) was held at the International Institute of Information Technology, Hyderabad, from December 15 to December 17, 2012. This proceedings volume contains contributed papers and (extended) abstracts of invited talks presented at the conference.

FSTTCS is the annual flagship conference of the Indian Association for Research in Computing Science (IARCS). Over the past 32 years, the conference has consistently attracted top-quality submissions, and the legacy continued this year as well. We wish to thank all authors of submitted papers for helping make FSTTCS a competitive conference.

The selection of papers appearing in this volume owes a lot to the program committee consisting of 29 distinguished colleagues spread over seven countries. The program committee spent over two months evaluating the strengths and weaknesses of the 121 submissions received. In this, we received generous help from 183 external reviewers who together produced a total of 394 reviews. Every paper that was found within the scope of the conference was reviewed by at least 3 experts. After several rounds of elimination, we have put together a program consisting of 43 contributed papers and five invited talks. We are very grateful to the members of the program committee for contributing their valuable time for helping us in the critical task of selecting papers. We also thank all external reviewers for their detailed comments and critical assessment of papers.

In addition to the contributed works, the program featured talks by five eminent computer scientists. We are very grateful to Dimitris Achlioptas (University of Athens), Mikołaj Bojańcyzk (Institute of Informatics, Warsaw University), Patrice Godefroid (Microsoft Research, Redmond), Madhusudan Parthasarathy (University of Illinois at Urbana-Champaign) and Yuval Rabani (The Hebrew University of Jerusalem) for agreeing to travel to the conference and share their insights.

Paper submission, reviewing and program committee discussions for the conference were managed using EasyChair. We wish to thank the EasyChair team for making this excellent service available to us, and for ensuring that the reviewing, discussion and paper management processes went smoothly.

As with the past editions of the conference, FSTTCS 2012 has interesting associated events. We are grateful to K. Narayan Kumar and Anil Seth for helping us organize the post-conference workshop on Verification of Infinite-State Systems. We also thank the speakers who agreed to participate in the workshop: Mohamed Faouzi Atig, Alain Finkel, Gennaro Parlato, Nishant Sinha, and Parosh Aziz Abdulla. In addition, we are grateful to Ravi Kannan, Satya Lokam and P.J. Narayanan for organising the joint MSR-IIIT winter school on Theoretical Computer Science, and thank the speakers at the winter school: Sanjeev Arora, Cnythia Dwork, Ravi Kannan, and R. Ravi.

We would like to thank IIIT Hyderabad for hosting FSTTCS 2012. We gratefully acknowledge the immense efforts of the administration, faculty, staff, and students of IIIT Hyderabad, especially the members of the organizing committee, over the past year in planning, coordinating and running various aspects of the conference, including managing the accounts, network services, video recordings, and the logistics. We are grateful for the financial support the conference received from Google India Pvt. Ltd.

As in the past few years, the proceedings of FSTTCS 2012 is being published as a volume in the LIPIcs series under a Creative Commons license, with free online access

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, T. Kavitha, and J. Radhakrishnan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to all, and with authors retaining rights over their contributions. We wish to thank the editorial board of LIPIcs for agreeing to publish the current proceedings as a LIPIcs volume. Marc Herbstritt and Michael Wagner (Dagstuhl) responded promptly and accurately to our questions pertaining to publishing and editing. We gratefully acknowledge their help and the help of the entire team in the LIPIcs editorial office in preparing the final version of the proceedings.

Deepak D'Souza, Telikepalli Kavitha and Jaikumar Radhakrishnan
December 2012

Conference Organization

Program Chairs

Deepak D'Souza (IISc, Bangalore, India)
Telikepalli Kavitha (TIFR, Mumbai, India)
Jaikumar Radhakrishnan (TIFR, Mumbai, India)

Program Committee

Nikhil Bansal (TU Eindhoven, The Netherlands)
Surender Baswana (IIT Kanpur, India)
Sourav Chakraborty (CMI, Chennai, India)
Thomas Colcombet (LIAFA, France)
Amit Deshpande (MSR, Bangalore, India)
Volker Diekert (U. of Stuttgart, Germany)
Chien-Chung Huang (Humboldt-U., Germany)
Rahul Jain (NUS, Singapore)
Kishore Kothapalli (IIIT Hyderabad, India)
Stefan Kratsch (U. of Utrecht, The Netherlands)
Steve Kremer (INRIA Nancy, LORIA, France)
Marta Kwiatkowska (University of Oxford, UK)
Christof Loeding (RWTH, Aachen, Germany)
Rupak Majumdar (MPI-SWS, Germany)
Oded Maler (VERIMAG, Grenoble, France)
Nicolas Markey (LSV, Cachan, France)
Aranyak Mehta (Google Research, USA)
Madhavan Mukund (CMI, Chennai, India)
Anca Muscholl (LABRI, Bordeaux, France)
Sanjiva Prasad (IIT Delhi, India)
Harald Raecke (TU Munich, Germany)
G. Ramalingam (MSR, Bangalore, India)
Saurabh Ray (MPI, Saarbruecken, Germany)
Rahul Santhanam (U. of Edinburgh, UK)
Shubhangi Saraf (Rutgers University, USA)
Anil Seth (IIT Kanpur, India)
Nikhil Srivastava (MSR, Bangalore, India)
Wolfgang Thomas (RWTH, Aachen, Germany)
Mahesh Viswanathan (UIUC, USA)

Organizing Committee

R K Bagga (IIIT Hyderabad), Chair
Bruhadeshwar Bezawada (IIIT Hyderabad)
Ashok Kumar Das (IIIT Hyderabad)
Kishore Kothapalli (IIIT Hyderabad)
P J Narayanan (IIIT Hyderabad)
Suresh Purini (IIIT Hyderabad)

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, T. Kavitha, and J. Radhakrishnan

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

External Reviewers

Samy Abbes
S. Akshay
Yoram Bachrach
Christel Baier
Bruno Blanchet
Patricia Bouyer
Arnaud Carayol
Krishnendu Chatterjee
Yu-Fang Chen
Andrea Corradini
Sanjoy Dasgupta
Aldric Degorre
Marco Diciolla
Vijay D'Silva
Bruno Escoffier
Diego Figueira
Dmitry Gavinsky
Inge Li Gørtz
Manoj Gupta
Matthias Hellwig
Bjarki Holm
Hiroshi Imai
Mark Jones
Prateek Karandikar
Jonathan Kausch
Eun Jung Kim
Barbara König
Sebastian Kuhnert
Nirman Kumar
Dietrich Kuske
Peeter Laud
Ranko Lazic
Nutan Limaye
Michael Luttenberger
Paul-André Melliès
Roland Meyer
Raj Mohan M.
Partha Mukhopadhyay
Ashwin Nayak
Fidaa Abed
Antonios Antoniadis
Michael Backes
Vince Barany
Benedikt Bollig
Thomas Brihaye
Rohit Chadha
Avik Chaudhuri
Corina Cirstea
Alfredo Costa
Samir Datta
Stephanie Delaune
Evan Driscoll
Michael Elberfeld
Shahram Esmaeilsabzali
Vojtech Forejt
Marcus Gelderie
Navin Goyal
Arie Gurfinkel
Ulrich Hertrampf
Florian Horn
Gabor Ivanyos
Hossein Jowhari
Hrishikesh Karmarkar
Rohit Khandekar
Felix Klaedtke
Guy Kortsarz
Raghav Kulkarni
Denis Kuperberg
Ugo Dal Lago
Fribourg Laurent
Troy Lee
Didier Lime
Bas Luttik
Alexandru Mereacre
Martin Milanic
Benjamin Monmege
S Muthukrishnan
Daniel Neider
Luca Aceto
Mario S. Alvim
Amitabha Bagchi
Ulrich Berger
Ahmed Bouajjani
Karl Bringmann
Timothy M. Chan
Taolue Chen
Lorenzo Clemente
Marek Cygan
Laure Daviaud
Yuxin Deng
Andrew Drucker
Michael Elkin
Dan Feldman
Pierre Ganty
Stefan Gölle
Guy Gueta
Ichiro Hasuo
Alexander Heussner
Falk Hueffner
Nils Jansen
Lukasz Kaiser
Alexander Kartzow
Stefan Kiefer
Hartmut Klauck
Manfred Kufleitner
Amit Kumar
Joy Kuri
Kim Guldstrand Larsen
Alexander Lauser
Pascal Lenzner
Kamal Lodaya
Amaldev Manuel
Antoine Meyer
Pranabendu Misra
Larry Moss
Meghana Nasre
Dejan Nickovic

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, T. Kavitha, and J. Radhakrishnan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Prajakta Nimbhorkar
Petr Novotný
Dömötör Pálvölgyi
Achim Passen
Valentin Polishchuk
Suresh Purini
Rajiv Raman
Heiko Röglin
Prakash Saivasan
Pierre-Yves Schobbens
Rahul Shah
Michal Skrzypczak
Pierre-Yves Strub
Xiaoming Sun
Till Tantau
Ashish Tiwari
Michael Ummels
Enrico Vicario
Tobias Walter
John Watrous
Armin Weiss
Christian Wulff-Nilsen

Aditya Nori
Alfredo Olivero
Wied Pakusa
Marcin Pilipczuk
Vinod M Prabhakaran
Hongyang Qu
Venkatesh Raman
Sambuddha Roy
Ocan Sankur
Pranab Sen
Gautham Shenoy R
Aravind Srinivasan
Ondrej Suchy
S P Suresh
Tachio Terauchi
Hans Raj Tiwary
Dominique Unruh
Jules Villard
Igor Walukiewicz
Jeremias Weihmann
Ronald de Wolf
Shengyu Zhang

Gethin Norman
Friedrich Otto
David Parker
George Pierrakos
Gabriele Puppis
Kaushik Rajan
Klaus Reinhardt
Andrey Rybalchenko
Nitin Saurabh
Chintan Shah
Sunil Simon
Jan Strejcek
Andrew Suk
Jacint Szabo
P S Thiagarajan
Kyle Treleaven
Elad Verbin
Walter Vogler
Xu Wang
Pascal Weil
James Worrell
Florian Zuleger

Learning Mixtures of Distributions over Large Discrete Domains

Yuval Rabani[1]

1 **The Rachel and Selim Benin School of Computer Science and Engineering The Hebrew University of Jerusalem Jerusalem 91904, Israel**
yrabani@cs.huji.ac.il

Abstract

We discuss recent results giving algorithms for learning mixtures of unstructured distributions.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, I.5.2 Design Methodology

Keywords and phrases machine learning, mixture models, topic models

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.1

Summary

The past decade or so has witnessed tremendous progress in the theory of learning statistical mixture models. The most striking example is that of learning mixtures of high dimensional Gaussians. Starting from Dasgupta's ground-breaking paper [14], a long sequence of improvements [15, 5, 27, 21, 1, 17, 8] culminated in the recent results [20, 7, 23] that essentially resolve the problem in its general form. In this vein, other highly structured mixture models, such as mixtures of discrete product distributions [22, 19, 12, 18, 9, 11] and similar models [12, 6, 24, 21, 13, 10, 16], have been studied intensively.

Here we discuss recent results giving algorithms for learning mixtures of *unstructured* distributions. More specifically, we consider the problem of learning mixtures of k arbitrary distributions over a large discrete domain $[n] = \{1, 2, \ldots, n\}$. This problem arises in various unsupervised learning scenarios, for example in learning *topic models* from a corpus of documents spanning several topics. We discuss the goal of learning the probabilistic model that is hypothesized to generate the observed data, in particular the constituents (each topic distribution) of the mixture. It is information-theoretically impossible to reconstruct the mixture model from single-view samples (e.g., single word documents). Thus, multi-view access is necessary. It is desirable to minimize the *aperture* or number of views in each sample point, as well as the number of sample points needed, as these parameters govern both the applicability of an algorithm and its computational complexity. We will survey some of the results in recent papers [4, 2, 3], as well as our joint work with L.J. Schulman and C. Swamy [25, 26]. In particular, we will discuss some of the tools that contribute to these results, in brief: concentration results for random matrices, SVD and other factorizations, dimension reduction, moment estimations, and sensitivity analysis.

References

1 D. Achlioptas and F. McSherry. On spectral learning of mixtures of distributions. In *Proc. of the 18th Ann. Conf. on Learning Theory*, pages 458–469, June 2005.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 1–3

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

2 A. Anandkumar, D.P. Foster, D. Hsu, S.M. Kakade, and Y.-K. Liu. Two SVDs suffice: Spectral decompositions for probabilistic topic modeling and latent Dirichlet allocation. *CoRR*, abs/1204.6703, 2012.

3 A. Anandkumar, D. Hsu, and S.M. Kakade. A method of moments for mixture models and hidden Markov models. *CoRR*, abs/1203.0683, 2012.

4 S. Arora, R. Ge, and A. Moitra. Learning topic models — going beyond SVD. *CoRR*, abs/1204.1956, 2012.

5 S. Arora and R. Kannan. Learning mixtures of separated nonspherical Gaussians. *Ann. Appl. Probab.*, 15(1A):69–92, 2005.

6 T. Batu, S. Guha, and S. Kannan. Inferring mixtures of Markov chains. In *Proc. of the 17th Ann. Conf. on Learning Theory*, pages 186–199, July 2004.

7 M. Belkin and K. Sinha. Polynomial learning of distribution families. In *Proc. of the 51st Ann. IEEE Symp. on Foundations of Computer Science*, pages 103–112, October 2010.

8 S.C. Brubaker and S. Vempala. Isotropic PCA and affine-invariant clustering. In *Proc. of the 49th Ann. IEEE Symp. on Foundations of Computer Science*, pages 551–560, October 2008.

9 K. Chaudhuri, E. Halperin, S. Rao, and S. Zhou. A rigorous analysis of population stratification with limited data. In *Proc. of the 18th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 1046–1055, January 2007.

10 K. Chaudhuri and S. Rao. Beyond Gaussians: Spectral methods for learning mixtures of heavy-tailed distributions. In *Proc. of the 21st Ann. Conf. on Learning Theory*, pages 21–32, July 2008.

11 K. Chaudhuri and S. Rao. Learning mixtures of product distributions using correlations and independence. In *Proc. of the 21st Ann. Conf. on Learning Theory*, pages 9–20, July 2008.

12 M. Cryan, L. Goldberg, and P. Goldberg. Evolutionary trees can be learned in polynomial time in the two state general Markov model. *SIAM J. Comput.*, 31(2):375–397, 2002.

13 A. Dasgupta, J. Hopcroft, J. Kleinberg, and M. Sandler. On learning mixtures of heavy-tailed distributions. In *Proc. of the 46th Ann. IEEE Symp. on Foundations of Computer Science*, pages 491–500, October 2005.

14 S. Dasgupta. Learning mixtures of Gaussians. In *Proc. of the 40th Ann. Symp. on Foundations of Computer Science*, pages 634–644, October 1999.

15 S. Dasgupta and L.J. Schulman. A probabilistic analysis of EM for mixtures of separated, spherical Gaussians. *Journal of Machine Learning Research*, 8:203–226, 2007.

16 C. Daskalakis, I. Diakonikolas, and R.A. Servedio. Learning k-modal distributions via testing. In *Proc. of the 23rd Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 1371–1385, January 2012.

17 J. Feldman, R. O'Donnell, and R.A. Servedio. PAC learning mixtures of axis-aligned Gaussians with no separation assumption. In *Proc. of the 19th Ann. Conf. on Learning Theory*, pages 20–34, June 2006.

18 J. Feldman, R. O'Donnell, and R.A. Servedio. Learning mixtures of product distributions over discrete domains. *SIAM J. Comput.*, 37(5):1536–1564, 2008.

19 Y. Freund and Y. Mansour. Estimating a mixture of two product distributions. In *Proc. of the 12th Ann. Conf. on Computational Learning Theory*, pages 183–192, July 1999.

20 A.T. Kalai, A. Moitra, and G. Valiant. Efficiently learning mixtures of two Gaussians. In *Proc. of the 42nd Ann. ACM Symp. on Theory of Computing*, pages 553–562, June 2010.

21 R. Kannan, H. Salmasian, and S. Vempala. The spectral method for general mixture models. *SIAM J. Comput.*, 38(3):1141–1156, 2008.

22 M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. On the learnability of discrete distributions. In *Proc. of the 26th Ann. ACM Symp. on Theory of Computing*, pages 273–282, May 1994.

23 A. Moitra and G. Valiant. Settling the polynomial learnability of mixtures of Gaussians. In *Proc. of the 51st Ann. IEEE Symp. on Foundations of Computer Science*, pages 93–102, 2010.

24 E. Mossel and S. Roch. Learning nonsingular phylogenies and hidden Markov models. In *Proc. of the 37th Ann. ACM Symp. on Theory of Computing*, pages 366–375, May 2005.

25 Y. Rabani, L.J. Schulman, and C. Swamy. Inference from sparse sampling. http://www.cs.technion.ac.il/~rabani/Papers/RabaniSS-manuscript.pdf, 2008.

26 Y. Rabani, L.J. Schulman, and C. Swamy. Learning mixtures of arbitrary distributions over large discrete domains. Unpublished, 2012.

27 S. Vempala and G. Wang. A spectral algorithm for learning mixtures of distributions. *J. Computer and System Sciences*, 68(4):841–860, 2004.

Imperative Programming in Sets with Atoms*

Mikołaj Bojańczyk and Szymon Toruńczyk

University of Warsaw, Warsaw, Poland

Abstract

We define an imperative programming language, which extends while programs with a type for storing atoms or hereditarily orbit-finite sets. To deal with an orbit-finite set, the language has a loop construction, which is executed in parallel for all elements of an orbit-finite set. We show examples of programs in this language, e.g. a program for minimising deterministic orbit-finite automata.

1998 ACM Subject Classification D.3.3 Language Constructs and Features, F.4.1 Mathematical Logic

Keywords and phrases Nominal sets, sets with atoms, while programs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.4

Introduction

This paper introduces a programming language that works with sets with atoms, which appear in the literature under various other names: Fraenkel-Mostowski models [2], nominal sets [7], sets with urelements [1], permutation models [9].

Sets with atoms are an extended notion of a set – such sets are allowed to contain "atoms". The existence of atoms is postulated as an axiom. The key role in the theory 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. In a more general setting, the atoms have some structure, and instead of permutations one talks about automorphisms of the atoms. Suppose for instance that the atoms are real numbers, equipped with the successor relation $x = y + 1$ and linear order $x < y$. Then the sets

$$\{-1, 0, 0.3\} \qquad \{5.2, 6.2, 6.12\}$$

are equal up to automorphism of the atoms, but the sets

$$\{0, 2\} \qquad \{5.3, 8.3\}$$

are not.

Here is the definition of sets with atoms. The definition is parametrized by a notion of atoms. The atoms are given as a relational structure, which induces a notion of automorphism. (One can also consider atoms with function symbols, but we do not do this here.) A *set with atoms* is any set that can contain atoms or other sets with atoms, in a well-founded way[1]. The key notion is the notion of a *legal* set of atoms, defined below. Suppose that

* Supported by ERC Starting Grant "Sosna".

[1] Formally speaking, sets with atoms are defined by induction on their rank, which is an ordinal number. Sets of a given rank can contain atoms and sets of lower rank.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 4–15

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

X is a set with atoms. If π is an automorphism of atoms, then π can be applied to X, by renaming all atoms that appear in X, and appear in elements of X, and so on. We say that a set S of atoms is a *support* of the set X if X is invariant under every automorphism of atoms which is the identity on S. (For instance, the set of all atoms is supported by the empty set, because every automorphism maps the set to itself.) A set with atoms is called *legal* if it has some finite support, each of its elements has some finite support, and so on recursively.

Sets with atoms were introduced in set theory by Fraenkel in 1922. Fraenkel gave a set of axioms for set theory, call it Zermelo-Fraenkel with Atoms (ZFA), which is similar but not identical to the standard axioms of Zermelo-Fraenkel (ZF). One difference is that ZFA does not have the extensionality axiom: two objects (think of different atoms) might have the same elements, but not be equal. Fraenkel gave two models of ZFA: one which has the axiom of choice, and one which does not. The first model contains all sets with atoms, while the second model restricts to the legal ones. Legal sets with atoms were further developed by Mostowski, which is why they are sometimes called Fraenkel-Mostowski sets.

In this paper, we are exclusively interested in sets with atoms that are legal. Therefore, from now on all sets with atoms are assumed to be legal.

Sets with atoms (as remarked above, we implicitly restrict to legal ones) were rediscovered for the computer science community, by Gabbay and Pitts [7]. In this application area, atoms have no structure, and therefore automorphisms are arbitrary permutations of atoms. It turns out that atoms are a good way of describing variable names in programs or logical formulas, and the automorphisms of atoms are a good way of describing renaming of variables. Sets with atoms are now widely studied in the semantics community, under the name of nominal sets (the name is so chosen because atoms describe variables names).

Sets with atoms turn out to be a good framework for other applications in computer science. These other applications have roots in database theory, but touch 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. 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.

Sets with atoms are a good abstraction for infinite systems because they have a different, more relaxed, notion of finiteness. A set with atoms is considered finite if it has finitely many elements, up to automorphisms of atoms. (The formal definition is given later in the paper.) We call such a set *orbit-finite*. Consider for example sets with atoms where the atoms have no structure, and therefore automorphisms are arbitrary permutations. The set of atoms itself is orbit-finite, actually has only one orbit, 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, with variables being atoms:

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

this set has one orbit. Yet another example concerns automata with registers for storing atoms, as introduced by Francez and Kaminski in [10]: up to permutation, there are finitely many configurations of every such automaton.

The language of sets with atoms is so robust that one can meaningfully restate all definitions and theorems of discrete mathematics replacing sets by (legal) sets with atoms and finite sets by orbit-finite sets, see [3, 5] for examples in automata theory. Some of the

restated theorems are true, some are not. Results that fail after adding atoms include all results which depend on the subset construction, such as determinisation of finite automata, or equivalence of two-way and one-way finite automata. Results that work after adding atoms include the Myhill-Nerode theorem, or the equivalence of pushdown automata with context free grammars (under certain assumptions on the structure of atoms, which will be described below).

The papers [3, 5] were concerned with generalisations of finite state machines, like finite monoids and finite automata. But what about general computer programs? Is there are notion of computability for sets with atoms? One attempt at answering this question was [4], which described a functional programming language equipped with types for storing orbit-finite sets. The present paper gives an alternative answer: an imperative programming language, called while programs with atoms[2].

What is the advantage of having a programming language which can handle sets with atoms, be it functional or imperative? Consider the following algorithmic tasks coming from automata theory:

1. Given a nondeterministic finite automaton, decide if it is nonempty.
2. Given a deterministic finite automaton, compute the minimal automaton.
3. Given a finite monoid, decide if it is aperiodic.
4. Given a context-free grammar, compute an equivalent pushdown automaton.

Each of these tasks can be studied in the presence of atoms. (Finite automata with atoms are defined in [5], finite monoids with atoms are defined in [3], context-free grammars and pushdown automata can be defined in the same spirit.) Without a computation model of some sort, it is not clear what it means that the above tasks are decidable. In the papers [3, 5], the computational model depended on coding: a finite automaton with atoms was encoded as a normal string over the alphabet $\{0, 1\}$, and then the remainder of the algorithm used a standard Turing machine. Such coding is not a satisfactory solution, since algorithms and correctness proofs that involve coding are tedious and error-prone.

The picture becomes much simpler when using a language that manipulates directly objects with atoms. The coding issues have to be dealt with only once; when designing the language and proving that its programs can be simulated by usual computers. What is also interesting, if the syntax of the programming language is based on a classical syntax without atoms (in this paper, we add atoms to while programs), then one can easily compare programs for the same task with and without atoms. For instance, in the four tasks described above, one can write a program with atoms that actually has the same code as the corresponding program without atoms, only the interpreter used to execute it has different semantics.

1 Orbit-finite sets with atoms

In this section, we define orbit-finite sets with atoms. The definition of an orbit-finite set can be stated for any notion of atoms (modelled as a relational structure). However, without additional assumptions on the atoms, the notion of orbit-finite set might not be well behaved, e.g. orbit-finite sets might not be closed under products or finitely supported subsets. An assumption that guarantees good behaviour of orbit-finite sets is called homogeneity, and is defined below.

[2] We believe that the two programming languages are equivalent, under a suitable encoding, but we do not prove this in the paper.

Homogeneous structures

Recall that the notion of a set with atoms is parametrized by a relational structure for the atoms. Examples of atoms are:

$(\mathbb{N}, =)$ natural numbers (or any countably infinite set) with equality

$(\mathbb{Q}, \leq)$ the rational numbers with their order

The two kinds of atoms listed above will be called, respectively, the *equality atoms* and the *total order atoms*. In this paper, we require the atoms to be a *homogeneous* relational structure, i.e. one which satisfies the following property:

> Any isomorphism between two finite substructures of the atoms extends to an automorphism of the atoms.

Moreover, we assume that the atoms are countable, and the vocabulary of the relational structure is finite. The programming language we describe in this paper will work with any atoms satisfying the conditions above, plus an additional decidability condition to be defined below[3].

The equality and total order atoms are countable, homogeneous and have a finite vocabulary. An example of a structure which is not homogenous is $(\mathbb{Z}, \leq)$, the set of integers with their order. Indeed, the subsets $\{1, 2\}$ and $\{1, 3\}$ induce isomorphic substructures of $(\mathbb{Z}, \leq)$; however, there is no automorphism of $\mathbb{Z}$ which maps 1 to 1 and 2 to 3. In the rest of the paper, we always assume that the atoms are a countable homogeneous structure.

An interesting example of a homogeneous structure is the *random (undirected) graph*, also called the Rado graph.

▶ Example 1 (Random graph). The universe of this structure – representing the vertices of the graph – is the set of natural numbers. There is one binary relation, representing the edges of the graph; it is symmetric and irreflexive. The edges are constructed as follows: independently for each pair of vertices v, w, with probability $1/2$ we declare that v and w are connected by an edge. One can prove that two graphs constructed as above are isomorphic with probability 1; which is why we talk about *the* random graph and not *some* random graph. Moreover, the random graph is homogeneous.

An important property of homogeneous structures is that finitely supported relations coincide with sets of atoms definable by quantifier-free formulas which can use constants from the atoms. More precisely, we have the following.

▶ Proposition 1. Assume that the atoms are a homogeneous structure over a finite vocabulary. Let S be a finite set of atoms, and R a set of n-tuples of atoms. Then, R is S-supported if and only if it is defined by a quantifier-free formula over the vocabulary of the atoms, extended by constant names for elements of S.

For a given S and n, there are finitely many quantifier-free formulas over n-variables which use the (finite) vocabulary and constants from S. From the above proposition it follows that there are finitely many S-supported sets of n-tuples of atoms, for any given S and n.

[3] With slightly more work, the programming language would also work for the more general notion of atoms that are *oligomorphic* or, equivalently, *ω-categorical*.

Hereditarily orbit-finite sets

The reason why we are interested in sets with atoms is that there is an interesting new notion of finiteness, which is described below. Recall that for each legal set X there is a finite support S, i.e. the set X is invariant under the action of automorphisms which are the identity over S. We call such automorphisms *S-automorphisms.* Stated equivalently, X is a union of *S-orbits*, i.e. equivalence classes of the following relation $\sim_S$:

$$x \sim_S y \qquad \text{if } \pi \cdot x = y \text{ for some } S\text{-automorphism } \pi.$$

We say that X is *orbit-finite* if the union is finite, i.e. X is union of finitely many S-orbits for some finite set of atoms S. One of the important properties of homogeneous atoms is that the definition of orbit-finiteness does not depend on the choice of support S:

▶ **Lemma 2.** *Let X be a (legal) set, which is supported by sets S and T. Then X has finitely many S-orbits if and only if it has finitely many T-orbits.*

Other advantages of homogeneous atoms include:

- For every $n \in \mathbb{N}$, every finitely supported subset of n-tuples of atoms is orbit-finite;
- Orbit-finite sets are closed under products and finitely supported subsets.

In our programming language, we deal with sets that are *hereditarily orbit-finite*, i.e. sets which are orbit-finite, whose elements are orbit-finite, and so on recursively until an atom or empty set is reached. (One can show that for every hereditarily orbit-finite the nesting of set brackets is a natural number, as opposed to other sets, where the nesting may be an ordinal number.) As we will show, such sets can be presented in a finite way, and manipulated using algorithms.

Decidable homogeneous structures

By the theorem of Fraïssé [6], a homogenous structure is determined uniquely (up to isomorphism) by its *age*, which is the class of structures that embed into it. To permit computation, we will require an effective representation of this family.

▶ **Definition 3** (Decidable homogeneous structure)**.** A homogeneous structure $\mathfrak{A}$ is called *decidable* if its vocabulary is finite and its age is decidable, i.e. one can decide whether a given finite structure embeds into $\mathfrak{A}$.

The equality atoms and the total order atoms are decidable: the age of the equality atoms is all finite structures over an empty vocabulary, while the age of the total order atoms is all finite total orders. By Fraïssé's theorem, an algorithm deciding the age defines uniquely (up to isomorphism) the structure of Atoms. It follows that there are, up to isomorphism, countably many decidable homogenous structures. By [8], there are uncountably many non-isomorphic homogeneous structures over the signature containing one binary symbol, so some of them are undecidable.

The point of considering hereditarily orbit-finite sets in decidable homogeneous structures is that they can be represented without atoms, and these representations can be manipulated by algorithms.

▶ **Theorem 4.** *Suppose that the atoms are a decidable homogeneous structure over a finite vocabulary. Then there are data structures for representing atoms and hereditarily orbit-finite sets, which admit the following operations:*

1. *Given atoms $a_1, \ldots, a_n$ and a n-ary relation R from the vocabulary of the atom structure, decide if $R(a_1, \ldots, a_n)$ holds;*
2. *Given hereditarily orbit-finite sets X, Y, decide if $X = Y$;*
3. *Given X, which is a hereditarily orbit-finite set or an atom, compute $\{X\}$;*
4. *Given hereditarily orbit-finite sets X, Y, compute $X \cup Y$;*
5. *Given a hereditarily orbit-finite set X, compute some finite support;* [4]
6. *Given a hereditarily orbit-finite set X and a finite set S of atoms, produce all S-orbits that intersect X;* [5]
7. *Decide if a hereditarily orbit-finite set X is empty. If it is nonempty, produce an element.*

For instance, under the equality atoms, an atom can be represented as a natural number, encoded as its binary representation. The total order atoms are rational numbers, so they can also be represented, and the order relation can be computed. The representation for hereditarily orbit-finite sets is more involved.

2 Imperative programming with atoms

In this section, we present the contribution of the paper, an imperative programming language with atoms. The language extends while programs with two types: one for storing atoms and one for storing hereditarily orbit-finite sets. To deal with an orbit-finite set, the language has a loop construction, which is executed in parallel for all elements of an orbit-finite set. We end the paper with examples of programs in this language, e.g. a program for minimising deterministic orbit-finite automata.

2.1 Definition of the imperative language

The language is called *while programs with atoms*. The definition of the language depends on the choice of atoms (but not too much). We assume that the atoms are a decidable homogeneous relational structure over a finite vocabulary, as in the assumptions of Theorem 4.

The datatype.

We only have two datatypes in our language: `atom` and `set`. A variable of type `atom` stores an atom or is *undefined*. A variable of type `set` stores a hereditarily orbit-finite set. To have a minimal language, we encode other types inside `set`, using standard set-theoretic encodings. For example, the boolean `true` is encoded by $\{\emptyset\}$, and the boolean false is encoded by $\emptyset$.

Syntax.

The language contains the following constructions:

[4] This finite support is represented as a list of atoms.

[5] The S-orbits that intersect X are given as a list of sets. We claim that this list is finite. Indeed, the set X has some support, say T. Without loss of generality, we may assume that $S \subseteq T$, because supports are closed under adding elements. By assumption that X is orbit-finite and by Lemma 2, X is a finite union of T-orbits. Since $S \subseteq T$, every S-orbit is a union of T-orbits. It follows that X intersects at most finitely many S-orbits.

- **Constants.** There are infinitely many constants of type `atom`: one constant for every atom. (These constants depend on the choice of atom structure, e.g. there will be different constants for the equality atoms and different constants for the total order atoms.) There are constants $\emptyset$ and `Atoms` of type `set`, representing the empty set and the set of all atoms.
- **Expressions.** Expressions (which have values in the type `atom` or `set`), can be built out of variables and constants, using the following operations:
  1. Variables and constants are expressions. We assume that the types of the variables are declared in a designated preamble to the program; variables of type `atom` are initially undefined, while variables of type `set` are initially set to $\emptyset$.
  2. For every symbol σ in the vocabulary of the atom structure, if σ has arity n and $\mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_n$ are expressions of type `atom`, then
  $$\sigma(\mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_n)$$
  is an expression which evaluates to `true` or `false` (such an expression is of type `set`). For instance, when the atom structure is the total order atoms, and `x` and `y` are variables of type atom, then $\mathtt{x} \leq \mathtt{y}$ is an expression (we write $\leq$ using infix style).
  3. Comparisons $\mathtt{e} \in \mathtt{f}$ and $\mathtt{e} = \mathtt{f}$, which evaluate to `true` or `false`. In these comparisons, `e` and `f` can be either of type `atom` or `set`.
  4. Union $\mathtt{e} \cup \mathtt{f}$, intersection $\mathtt{e} \cap \mathtt{f}$ and set difference $\mathtt{e} - \mathtt{f}$, for `e` and `f` of type `set`.
  5. A unary singleton operation which adds one set bracket $\{\mathtt{e}\}$. Here, `e` can be either of type `atom` or `set`.
  6. An operation which extracts the unique atom from a set:
  $$\mathtt{theunique}(\mathtt{e}) = \begin{cases} \mathtt{f} & \text{when } \mathtt{e} = \{\mathtt{f}\} \text{ for some } \mathtt{f} \text{ of type } \mathtt{atom} \\ \mathtt{undefined} & \text{otherwise.} \end{cases}$$
- Values of expressions can be assigned to variables using the instruction $\mathtt{x} := \mathtt{e}$, provided that the types match.
- **Programming constructions.**
  1. A conditional `if e then I else J`. If the value of expression `e` is `true` $\stackrel{\text{def}}{=} \{\emptyset\}$, then program `I` is executed, otherwise program `J` is executed.
  2. A while loop `while e do I`, which executes the program `I`, while the value of expression `e` is `true`.
  3. A parallel for loop `for x in X do I`. Here `X` is an expression of type `set` and `x` is a variable of either type. The general idea is that the instruction `I` is executed, in parallel, with one thread for every element `x` (of appropriate type) of the set `X`. The question is: how are the results of the threads combined? We answer this question in more detail below.

Semantics.

We now sketch a semantics (operational style) for the language. In a given program, a finite number of variables is used. A state of the program is a valuation ν which assigns atoms (or the undefined value) to variables of type `atom` and hereditarily orbit-finite sets to variables of type `set`. Essentially, a valuation is a (finite length) tuple containing atoms and hereditarily orbit-finite sets, and therefore the set of all valuations is a legal set with atoms (but not orbit-finite). A state of the program can be represented in a finite way using the data structures from Theorem 4.

The semantics of a program is a partial function, which maps one valuation to another valuation. (The function is partial, because for some valuations, the program might not terminate.) We will say that *executing* the program `P` on the valuation ν results in a valuation μ if the semantics of the program transforms the valuation ν to the valuation μ.

We only explain the semantics for programs of the form

$$\texttt{for x in X do I},$$

the other semantics are defined in the standard way. Suppose that we want to execute the program on a valuation ν. Two cases need be considered: when `x` is a variable of type `set` or when `x` is variable of type `atom`. The set $\nu(\texttt{X})$ might store elements of both types `set` and `atom`. We say that an element $x \in \nu(\texttt{X})$ is *appropriate* if it matches the type of the variable `x`. We define the valuation resulting from executing the above instruction on the valuation ν as follows. For every appropriate $x \in \nu(\texttt{X})$, we execute the instruction `I` on the valuation ν_x which is obtained from the valuation ν by putting value x in the variable `x`. If for some appropriate x the program `I` does not terminate, then the whole `for` program does not terminate. Otherwise, for each appropriate $x \in \nu(\texttt{X})$, we get a valuation μ_x obtained from ν_x by executing `I`. We now want to aggregate the valuations μ_x into a single valuation μ, which will be the result of executing the `for` program. If `y` is a variable of type `atom`, then in order for $\mu(\texttt{y})$ to be defined, we require that all valuations agree on the value of `y`:

$$\mu(\texttt{y}) \stackrel{\text{def}}{=} \begin{cases} a & \text{if for all appropriate } x \in \nu(\texttt{X}),\ \mu_x(\texttt{y}) = a \\ \texttt{undefined} & \text{otherwise} \end{cases} \tag{1}$$

Set variables are aggregated using set union, i.e. every variable `y` of type `set` gets set to

$$\mu(\texttt{y}) = \bigcup_{\text{appropriate } x \in \nu(\texttt{X})} \mu_x(\texttt{y}). \tag{2}$$

The definition of the language is now complete.

Results

One can show that our semantics is well-defined, i.e. executing instructions of our programming language does not cause a set variable to be assigned a set that is not hereditarily orbit-finite. We also prove that the programs can be simulated without atoms, in the following way. Thanks to Theorem 4 the code of a program, as well as a valuation of the variables, can be represented in a finite way without atoms.

▶ **Theorem 5.** *There is a normal program (without) atoms* `P`*, which inputs:*

- *a while program with atoms* `I`*;*
- *a valuation ν of the variables that appear in* `I`*;*

represented using the data structures of Theorem 4, and does the following:

- *if* `I` *does not terminate when starting in ν, then also* `P` *does not terminate;*
- *if* `I` *terminates when starting in ν, reaching valuation μ, then also* `P` *terminates, and produces a representation of valuation μ.*

Since the representations do not use atoms, they can be seen as standard bit strings, i.e. words over the alphabet $\{0, 1\}$. Therefore `P` can be modelled as a Turing machine, which inputs two bit strings and outputs a single bit string (and possibly does not terminate). In the proof of the above theorem we use the properties of the representations which are listed in Theorem 4.

The rest of the paper is devoted to example programs.

2.2 Example programs

Before writing example programs, we introduce some syntactic sugar which makes programming easier.

Notational conventions

Like in Python, we use indentation to distinguish blocks in programs. We write `{x,y}` instead of `{{x}∪{y}}`. We extend the syntax with functions (with the usual semantics); the syntax of functions is illustrated on the Kuratowski pairing function

```
function pair(x,y)
  return {{x},{x,y}}
```

We write `(a,b)` instead of `pair(a,b)`. Here is the function which projects a Kuratowski pair of sets into its first coordinate, and returns $\emptyset$ if its argument is not a Kuratowski pair of sets. All the variables are assumed to be of type `set`.

```
function first(p)
  for a in p do
    for b in p do
      for x in a do
        for y in b do
          if p = {{x},{x,y}} then ret:=x;
  return ret
```

The second coordinate of a pair is extracted the same way. Similarly, we could write functions for projections of pairs storing atoms, or pairs storing one atom and one set. Using the projections, we can extend the language with a pattern-matching construction

```
for (x,y) in X do I
```

which ranges over all elements of `X` that are pairs of elements of appropriate types. We use a similar convention for tuples of length greater than two.

▶ Example 2 (The diagonal). As a warmup, we write a program that produces a specific set, namely

$$\{(a, a) : a \in \text{Atoms}\}.$$

The following program calculates this set in variable `X`.

```
for x in Atoms do X := X ∪ {(x,x)}
```

The same effect would be achieved by the following program.

```
for x in Atoms do X := {(x,x)}
```

▶ Example 3 (Programs that use order on atoms). In the same spirit, we can produce sets that refer to some structure on the atoms.

Consider the total order atoms. Recall that there is an expression `x` $\leq$ `y` that says if the atom stored in variable `x` is smaller than the atom stored in variable `y`. For instance, the following program generates the growing triples of atoms.

```
for x in Atoms do
 for y in Atoms do
  for z in Atoms do
    if (x ≤ y) and (y ≤ z) then X := {(x,y,z)}
```

▶ Example 4. Consider the total order atoms. The following program produces in variable `Y` the family of all closed intervals.

```
for (x,y) in Atoms do
   for z in Atoms do
      if (x ≤ z) and (z ≤ y) then X := X ∪ {z}
   Y := Y ∪ {X}
```

Actually, for every atom structure and for every hereditarily orbit-finite set X there exists a program which produces the set X.

▶ Example 5 (Reachability). We write a program which inputs a binary relation `R` and a set of source elements `S`, and returns all elements reachable (in zero or more steps) from elements in `S` via the relation `R`. The program is written using `until`, which is implemented by `while` in the obvious way.

```
function reach (R,S)
  New := S
  repeat
   Old := New
   for (x,y) in R do
     if x ∈ Old then New := Old ∪ {y}
  until Old = New
```

The program above is the standard one for reachability, without any modifications for the setting with atoms. Why is the program still correct in the presence of atoms?

Suppose that S is a finite set of atoms that supports both the relation `R` and the source set `S`. Let X be the set that contains `S` and every element that appears on either the first or second coordinate of a pair from `R`. The set X is easily seen to be supported by S and to have finitely many S-orbits. Let $X_1, X_2, \ldots, X_k$ denote the S-orbits of X. Therefore,

$$X = X_1 \cup X_2 \cup \cdots \cup X_k. \tag{3}$$

It is easy to see that after every iteration of the `repeat` loop, the values of both variables `New` and `Old` are subsets of X that are supported by S. Therefore the values of these variables are obtained by selecting some of the orbits listed in (3). In each iteration of the `repeat` we add some orbits, and therefore the loop can be iterated at most k times.

▶ Example 6 (Automaton emptiness). Following [5], we define an *orbit-finite nondeterministic automaton* the same way as a nondeterministic automaton, with the difference that all of the components (input alphabet, states, initial states, final states, transitions) are required to be hereditarily orbit-finite sets. Using reachability, it is straightforward to write an emptiness check for nondeterministic orbit-finite automata:

```
function emptyautomaton(A,Q,I,F,delta)
  for (p,a,q) in delta do
    R := R ∪ {(p,q)}
  return ∅ = (reach(R,I) ∩ F)
```

▶ Example 7 (Monoid aperiodicity). An *orbit-finite monoid* is a monoid where the carrier is a hereditarily orbit-finite set, and the graph of the monoid operation is a finitely supported. (It follows that the graph of the monoid operation is a hereditarily orbit-finite set, because hereditarily orbit-finite sets are closed under finitely supported subsets.) Such a monoid is called *aperiodic* if for every element m of the monoid, there is a natural number n such that

$$m^n = m^{n+1}. \tag{4}$$

In [3] it was shown that an orbit-finite monoid is aperiodic if and only if all of the languages it recognises are definable in first-order logic. The following program inputs a monoid (its carrier and the graph of the monoid operation) and returns true if and only if the monoid is aperiodic. The program simply tests the identity (4) for every element in the carrier.

```
function aperiodic (Carrier,Monop)
  for m in Carrier  do
    X:=∅
    new:=m
    repeat
       old:=new
       X:=X ∪ {old}
       new := Monop(old,m)
    until new ∈ X
    if new = old then ret := true else ret := false
  return ret
```

In the program above, the line `new := mult(old,m)` is actually syntactic sugar for a subroutine, which examines the graph of the multiplication operation `mult`, and finds the unique element `new` which satisfies `(old,m,new)` $\in$ `mult`.

In the program, the set `X` is used to collect consecutive powers $\mathtt{m}, \mathtt{m}^2, \mathtt{m}^3, \ldots$. To prove termination, one needs to show that this set is always finite, even if the monoid in question is not aperiodic. This is shown in [3].

Finally, the program relies on the particular encoding of the booleans: `true` is the nonempty set $\{\emptyset\}$, while `false` is $\emptyset$. If the for loop sets `ret` to `true` for some `m` in the carrier of the monoid, then the whole program will return `true`, since the aggregation operation is union, which behaves like $\vee$ for booleans.

▶ Example 8 (Automaton minimisation). The programs for automaton emptiness and monoid aperiodicity were for yes/no questions. We now present a program that transforms one automaton into another. An *orbit-finite deterministic automaton* is the special case of the nondeterministic one where there is one initial state, and the transition relation is a function. As shown in [5], such automata can be minimised. We describe the minimisation procedure using a program in our language, i.e. a function

```
function minimize(A,Q,q0,F,delta)
```

which inputs an orbit-finite deterministic automaton and returns the minimal automaton[6]. We assume that all states are reachable, the non-reachable states can be discarded using

[6] This program is particularly interesting when comparing the imperative programming language from this paper with the functional programming language from [4]. In [4], we were unable to correctly type a program for minimisation.

the emptiness procedure described above. We will also assume that all states are sets (and not atoms), so all variables in the code below are declared as `set`. The code is a standard implementation of Moore's minimisation algorithm. The only point of writing it down here is that the reader can follow the code and see that it works with atoms.

In a first step, we compute in the variable `equiv` the equivalence relation, which identifies states that recognise the same languages.

```
for p in Q do
 for q in Q do
   for a in A do
      R := R ∪ {((delta(p,a),delta(q,a)),(p,q))}

base := (F × (Q-F)) ∪ ((Q-F) × F)
equiv := (Q × Q) - reach(R,base)
return ∅ = reach(R,q0) ∩ F
```

(The code above uses $\times$, which is implemented using `for`.) For the states of the minimal automaton, we need the equivalence classes of the relation `equiv`, which are produced by the following code.

```
function classes (equiv)
  for (a,b) in equiv do
    for (c,d) in equiv do
       if a=c then class := class ∪ {c}
    ret := ret ∪ {class}
  return ret
```

The remaining part of the minimisation program goes as expected: the states are the equivalence classes, and the remaining components of the automaton are defined as usual.

References

1. Jon Barwise. *Admissible sets and structures*. Springer-Verlag, Berlin, 1975. An approach to definability theory, Perspectives in Mathematical Logic.
2. Jon Barwise, editor. *Handbook of Mathematical Logic*. Number 90 in Studies in Logic and the Foundations of Mathematics. North-Holland, 1977.
3. Mikolaj Bojanczyk. Data monoids. In *STACS*, pages 105–116, 2011.
4. Mikolaj Bojanczyk, Laurent Braud, Bartek Klin, and Slawomir Lasota. Towards nominal computation. In *POPL*, pages 401–412, 2012.
5. Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata with group actions. In *LICS*, pages 355–364, 2011.
6. Roland Fraïssé. *Theory of relations*, volume 145 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, revised edition, 2000. With an appendix by Norbert Sauer.
7. Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.
8. C. Ward Henson. A family of countable homogeneous graphs. *Pacific J. Math.*, 38:69–83, 1971.
9. Thomas Jech. *The Axiom of Choice*. North-Holland, 1973.
10. Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

Algorithmic Improvements of the Lovász Local Lemma via Cluster Expansion*

Dimitris Achlioptas[1] and Themis Gouleakis[2]

1 **Department of Informatics and Telecommunications, University of Athens Athens, Greece**
optas@di.uoa.gr

2 **Department of Electrical Engineering and Computer Science, M.I.T. Cambridge, MA, USA**
tgoule@mit.edu

Abstract

The Lovász Local Lemma (LLL) is a powerful tool that can be used to prove that an object having none of a set of bad properties exists, using the probabilistic method. In many applications of the LLL it is also desirable to explicitly construct the combinatorial object. Recently it was shown that this is possible using a randomized algorithm in the full asymmetric LLL setting [R. Moser and G. Tardos, 2010]. A strengthening of the LLL for the case of dense local neighborhoods proved in [R. Bissacot et al., 2010] was recently also made constructive in [W. Pegden, 2011]. In another recent work [B. Haupler, B. Saha, A. Srinivasan, 2010], it was proved that the algorithm of Moser and Tardos is still efficient even when the number of events is exponential. Here we prove that these last two contributions can be combined to yield a new version of the LLL.

1998 ACM Subject Classification G.2.1 Combinatorics; G.3 Probabilistic Algorithms

Keywords and phrases Probabilistic Method, Lovász Local Lemma, Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.16

1 Introduction

The Lovász Local Lemma (LLL) [4] states that if one has a collection $\mathcal{A}$ of "bad" events in a probability space and each event is independent of all but "a few" other events in $\mathcal{A}$, then the probability that none of the bad events occurs is bounded away from 0. (We will give a precise and constructive statement shortly.)

Over the years there have been numerous efforts [1, 2, 7, 11, 8] to make the LLL constructive, culminating with the recent breakthrough of Moser and Tardos [9]. Specifically, in [9] one assumes that there is a finite set of n mutually independent random variables $\mathcal{P} = \{P_i\}$ and that each event in $\mathcal{A}$ is determined by a subset of $\mathcal{P}$. Let the dependency graph $G(\mathcal{A})$ have the events of $\mathcal{A}$ as its vertices and let two events A_i, A_j be deemed adjacent, denoted as $A_i \sim A_j$, iff they share at least one variable, i.e., $\mathrm{vbl}(A_i) \cap \mathrm{vbl}(A_j) \neq \emptyset$. The inclusive neighborhood of each event consists of itself and its adjacent events in G. (While this setting is not quite as general as the original LLL setting, as shown in [6], it suffices for the vast majority of applications of the LLL.) The goal is to find an assignment of values

* Research supported by ERC IDEAS Starting Grant RIMACO and a Sloan Research Fellowship. Part of this work was done while the first author was at UC Santa Cruz and the second author was with the National Technical University of Athens.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 16–23

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

to the n variables in $\mathcal{P}$ such that none of the events in $\mathcal{A}$ occurs. To that end, Moser and Tardos [9] showed that the following astonishingly simple algorithm suffices.

ALGORITHM MT

- Initialize the variables in $\mathcal{P}$ by sampling from their product measure.
- While any event in $\mathcal{A}$ occurs, select any occurring event and resample its variables according to their product measure.

▶ **Theorem 1** ([9]). *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events with a dependency graph G determined by these variables. Let Γ_i denote the inclusive neighborhood of event A_i in G.*

If there exists $\mu : \mathcal{A} \to (0, +\infty)$ such that

$$\forall A_i \in \mathcal{A} : \Pr[A_i] \leq \mu(A_i) \cdot \left(\sum_{U \subseteq \Gamma_i} \prod_{B \in U} \mu(B) \right)^{-1}, \tag{1}$$

then the expected number of resamplings performed by the randomized algorithm MT is bounded by $\sum_{A_i \in \mathcal{A}} \mu(A_i) = O(|A|) = O(m)$.

Clearly, since the running time of the algorithm is finite, we can trivially conclude that whenever a collection $\mathcal{A}_i$ of events satisfies (1), i.e., the LLL conditions, there exists an assignment to the variables such that none of them occurs.

In [3], Bissacot et al. gave a non-constructive strengthening of the LLL using cluster-expansion methods from statistical physics. Specifically, they proved that in the hypothesis of the LLL, in the rhs of (1), one can replace the summation over all subsets $U \subseteq \Gamma_i$ with a summation over subsets forming independent sets in $G(\mathcal{A})$. This improvement is significant if the subgraphs induced by the vertex neighborhoods in G are dense. On the other hand, the improvement is vanishing if the dependency graph is triangle-free.

In [10], Pegden made the result of Bissacot et al. constructive, by proving a theorem identical to Theorem 1, except for the aforementioned change in the range of the summation.

In a different direction, Haeupler, Saha, and Srinivasan [5] gave a more careful analysis of the MT algorithm, establishing that under slightly stronger assumptions the running time of the algorithm is independent of the number of events. This speedup can be exponential in certain applications.

▶ **Theorem 2** ([5]). *In the setting of Theorem 1, assume that there is $\varepsilon > 0$ such that*

$$\forall A_i \in \mathcal{A} : \Pr[A_i] \leq (1 - \varepsilon)\mu(A_i) \cdot \left(\sum_{U \subseteq \Gamma_i} \prod_{B \in U} \mu(B) \right)^{-1}. \tag{2}$$

Then the expected number of resamplings performed by algorithm MT is $O(\varepsilon^{-1} n \log(n/\varepsilon))$.

1.1 Our Results

We prove that the improvements of [10] and [5] stated earlier can be combined, yielding the strongest form of the LLL known to us.

As in the setting of Moser and Tardos [9], let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by the variables in $\mathcal{P}$ and let $G = G(\mathcal{A})$ be the the dependency graph of the events. Recall that Γ_i denotes the inclusive neighborhood of (the vertex corresponding to) each event A_i in G.

▶ **Definition 3.** For each Γ_i, let I_i consist of all subsets of Γ_i that are independent in $G(\mathcal{A})$. Let also $S_i = I_i \cup \{I \cup \{A_i\} : I \in I_i\}$, i.e., S_i contains every independent set of Γ_i along with its version enhanced by the addition of A_i.

▶ **Theorem 4.** *If there exists $\mu : \mathcal{A} \to (0, +\infty)$ and $\varepsilon > 0$ such that*

$$\forall A_i \in \mathcal{A} : \Pr[A_i] \leq (1-\varepsilon)\mu(A_i) \cdot \left(\sum_{U \in I_i} \prod_{B \in U} \mu(B) \right)^{-1} , \tag{3}$$

then the expected number of resamplings performed by MT is $O\left(\varepsilon^{-1} n \log\left(\sum_{A_i \in \mathcal{A}} \mu(A_i)\right)\right)$.

By slightly strengthening the condition in (3) the bound on the running time can be replaced by an expression that, in most applications, is independent of the number of events.

▶ **Theorem 5.** *In the setting of Theorem 4, assume that (3) holds if the summation is extended over all sets $U \in S_i$ (rather than over all sets $U \in I_i$). Then the expected number of resamplings performed by MT is bounded by $O(\varepsilon^{-1} n \log(Z/\varepsilon))$, where*

$$Z = Z(\mu) = \sum_{A_i \in \mathcal{A}} \frac{\mu(A_i)}{1 + \mu(A_i)} .$$

Clearly, the bound on the number of resamplings of Theorem 5 is no greater than the bound in Theorem 4, since each term in the sum is now divided by $1 + \mu(A_i)$. It was further shown in [5], that for most applications $Z = O(n \log n)$, implying that the expected number of resamplings in the conclusion of Theorem 5 is $O(\varepsilon^{-1} n \log(n/\varepsilon))$.

Indeed, our approach shows that all results from [5], e.g., the results regarding the case $\varepsilon = 0$, hold under the relaxed conditions in which one sums only over independent sets (or over the sets S_i if one wants to replace $\sum \mu(A_i)$ with $Z(\mu)/\varepsilon$ in the running time). As the proof technique is identical for all cases, we only show here the proofs of Theorems 4 and 5.

2 Witness Trees

2.1 Definition

A witness tree W is a finite, rooted tree where each vertex is labelled by an event. The analysis of the algorithm in [9] was made by mapping each prefix of the sequence of resampled events to a witness tree. Specifically, let L be the sequence of resampled events and let $L(t) = [A_{i_1}, A_{i_2}, ..., A_{i_t}]$ be the first t resampled events. The witness tree $W_L(t)$ is a *labelled* tree constructed as follows.

- Initialize $W_L(t)$ to a single node r (the root) with label A_{i_t} and depth $d(r) = 0$.
- For s from $t-1$ down to 1 do:
  1. Seek the deepest node of $W_L(t)$ whose event (label) is adjacent to A_{i_s} in $G(\mathcal{A})$.
  2. If you find such a node b, then update $W_L(t)$ by adding a node v with label A_{i_s} as a child of b and let $d(v) = d(b) + 1$.
  3. If no such b exists, do nothing.

▶ Remark. If $t_1 \neq t_2$ then $W_L(t_1) \neq W_L(t_2)$. (To see this observe that the witness tree associated with the k-th resampling of A_i will be the only witness tree with A_i as the root and exactly k nodes with label A_i.)

Moser and Tardos [9] provided an upper bound for the expected number of resamplings by studying the size of the witness trees generated by executions of their algorithm. Specifically, they proved the following upper bound on the probability that a specific witness tree W is ever generated for a given dependency graph G.

▶ **Lemma 6** ([9]). *For any witness tree W, let $M = M(W)$ be the multiset containing the labels of the vertices of W. Let X_W be the indicator random variable that W occurs in L.* $\Pr[X_W = 1] \leq \prod_{i \in M} \Pr[A_i]$.

The above lemma is used in [9] to bound the expected number of resamplings as follows. Let T be the random variable equal to the number of resamplings and let W_i be the set of all possible witness tress with root A_i. Since $T = |L|$ and each fixed witness tree occurs at most once in L, we get

$$\mathbb{E}[T] = \mathbb{E}[|L|] = \mathbb{E}\left[\sum_{A_i \in \mathcal{A}} \sum_{W \in W_i} X_W\right] = \sum_{A_i \in \mathcal{A}} \sum_{W \in W_i} \mathbb{E}[X_W] = \sum_{A_i \in \mathcal{A}} \sum_{W \in W_i} \Pr[X_W = 1] \ . \quad (4)$$

2.2 Random generation via branching process

To bound the sum in (4), Moser and Tardos [9] defined the set of *proper* witness trees to consist of all rooted, labelled trees in which the children of each node have distinct labels. Note that the set of proper witness trees contains all witness trees that can be generated by algorithm MT, since there cannot be two nodes with the same label at the same depth of the tree (the node that was added later must have gone at a strictly larger depth, if nothing else as a child of the earlier node).

To bound the total probability of all proper witness trees with root A_i, they defined the Galton-Watson branching process below and proved that the probability the algorithm generates any particular such witness tree W, i.e., $\Pr[X_W = 1]$, is bounded by a constant C times the probability that W is generated by the branching process. Therefore, each of the m events A_i is expected to appear at most C times in L and, thus, the expected total number of resamplings is bounded by Cm.

- Let $g(A_i) = \mu(A_i)(1 + \mu(A_i))^{-1}$.
- Create a single node r (the root) with label A_i and depth $d(r) = 0$.
- Let $d = 0$.
- **Repeat**
  1. For each node A at depth d consider its neighboring events in the dependency graph. For each such event B, with probability $g(B)$ add to A a child node with label B.
  2. $d \leftarrow d + 1$.
- **until** there is no node at depth d.

The bound on $\Pr[X_W = 1]$ comes by first applying Lemma 6, then substituting the bounds on $\Pr[A_i]$ from the LLL conditions, and finally relating the resulting expression to the exact expression for the probability that W is generated by the branching process.

2.3 The improvement of Pegden

Observe that since witness trees for the MT algorithm are grown by adding each event to the deepest possible level, the nodes at every level are labelled by events that form an independent set in the dependency graph G. Using this observation, in [10], Pegden defined witness trees in which *sibling* nodes must have non-adjacent events as *strongly proper* and

modified the Galton Watson process above as follows. To generate the progeny of each node we generate progeny-samples as before, i.e., via an independent trial for each neighboring event, but continue to reject until we get a progeny that forms an independent set in G. For this modified branching process, Pegden proved the following.

▶ **Lemma 7** ([10]). *For any strongly proper witness tree W with root labelled A_i, the probability that the branching process described above produces W is equal to*

$$p_W \equiv \mu(A_i)^{-1} \prod_{A_j \in W} \mu(A_j) \left(\sum_{U \in I_j} \prod_{B \in U} \mu(B) \right)^{-1} . \tag{5}$$

Moser and Tardos [9] had proved a similar lemma, where the summation is over all subsets of vertices in Γ_j instead of just the independent sets $U \in I_j$.

3 Proof of Theorem 4

We first state without proof a lemma that gives an upper bound on the expectation of a random variable given an exponential tail bound.

▶ **Lemma 8.** *Let X be an integer-valued random variable such that for some $\varepsilon > 0$, $C > 1$, and all $i > 0$, $\Pr[X \geq i] \leq C(1-\varepsilon)^i$. Then $\mathbb{E}[X] \leq \varepsilon^{-1}(1 + \varepsilon + \log C)$.*

Our plan is to bound the expected number of resamplings of the most frequent event. We will do this by bounding the expected size of the largest witness tree and using the fact that if an event A is resampled r times, then the witness tree associated with its last resampling will contain at least r nodes, as it will have exactly r nodes with label A. So, an upper bound on the expected size of the largest witness tree gives an upper bound on the expected number of resamplings of the most frequent event. Multiplying by n then gives an upper bound on the expected total number of resamplings.

For any integer k, let $Y(k)$ be the random variable equal to the number of witness trees that occur and have size at least k. Also, let $W_i^s(k)$ denote the set of strongly proper witness trees with root A_i and size at least k. Since the only witness trees that can possibly occur are strongly proper,

$$Y(k) = \sum_{A_i \in \mathcal{A}} \sum_{W \in W_i^s(k)} X_W . \tag{6}$$

By Markov's inequality, the probability that there is at least one witness tree of size at least k is bounded by the expected number of trees of size at least k that occur. Therefore, the probability that the largest witness tree in L has size at least k is bounded by

$$\sum_{A_i \in \mathcal{A}} \sum_{W \in W_i^s(k)} \Pr[X_W = 1] .$$

Under the conditions in (3), for every witness tree W of size at least k, Lemma 6 implies

$$\begin{aligned} \Pr[X_W = 1] &\leq \prod_{A_j \in W} (1-\varepsilon) \cdot \mu(A_j) \left(\sum_{U \in I_j} \prod_{B \in U} \mu(B) \right)^{-1} \\ &\leq (1-\varepsilon)^k \prod_{A_j \in W} \mu(A_j) \left(\sum_{U \in I_j} \prod_{B \in U} \mu(B) \right)^{-1} , \end{aligned} \tag{7}$$

where the second inequality follows from the fact that W has size at least k.

Substituting (7) into (6) we get (8) below and, by Lemma 7 we get (9).

$$\mathbb{E}[Y(k)] \leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \sum_{W \in W_i^s(k)} \prod_{A_j \in W} \mu(A_j) \left(\sum_{U \in I_j} \prod_{B \in U} \mu(B) \right)^{-1} \tag{8}$$

$$= (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \mu(A_i) \sum_{W \in W_i^s(k)} p_W \tag{9}$$

$$\leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \mu(A_i) \ . \tag{10}$$

Thus,

$$\Pr\left[\max_{W \in L} |W| \geq k\right] \leq \mathbb{E}[Y(k)] \leq (1-\varepsilon)^k \sum_{A_i \in \mathcal{A}} \mu(A_i) \ .$$

Lemma 8 thus implies that the expected size of the largest witness tree is bounded by

$$\varepsilon^{-1} \left(1 + \varepsilon + \log \left(\sum_{A_i \in \mathcal{A}} \mu(A_i) \right) \right) \tag{11}$$

which, as mentioned, implies that the expected number of resamplings of the MT algorithm is $O\left(\varepsilon^{-1} n \log\left(\sum_{A_i \in \mathcal{A}} \mu(A_i)\right)\right)$.

4 Proof of Theorem 5

Recall that our assumption is that there exists a function $\mu : \mathcal{A} \to (0, +\infty)$ such that

$$\forall A_i \in \mathcal{A} : \Pr[A_i] \leq (1-\varepsilon)\mu(A_i) \cdot \left(\sum_{U \in S_i} \prod_{B \in U} \mu(B) \right)^{-1} \ . \tag{12}$$

The idea is to prove that for every function μ satisfying (12), there is another function μ' satisfying (12) with slack somewhat less than ε, but all of whose values are bounded by some constant depending on ε. Observe that if μ' satisfies (12) with any slack, then μ' also satisfies the condition of Theorem 4 with the same slack since in that condition we only sum over $I_i \subseteq S_i$. Thus armed with μ' we first apply Theorem 4 to get a bound on the expected number of resamplings in terms of $\sum \mu'(A_i)$ and then exploit the boundedness of μ' to get a bound on the number of resamplings that depends only on $Z(\mu'), n$ and ε.

Recall that each set S_i contains all subsets of Γ_i that form independent sets in G, i.e., $I_i \subseteq S_i$, along with each element of I_i augmented by A_i. Therefore,

$$\sum_{U \in S_i} \prod_{B \in U} \mu(B) = (1 + \mu(A_i)) \sum_{U \in I_i \setminus \{A_i\}} \prod_{B \in U} \mu(u) \ . \tag{13}$$

Substituting (13) into (12) we get that for each event A_i,

$$\Pr[A_i] \leq (1-\varepsilon) \cdot \frac{\mu(A_i)}{1+\mu(A_i)} \left(\sum_{U \in I_i \setminus \{A_i\}} \prod_{B \in U} \mu(u) \right)^{-1} \ . \tag{14}$$

▶ **Lemma 9.** *If (12) holds for $\mu : \mathcal{A} \to (0, +\infty)$, then there is $\mu' : \mathcal{A} \to (0, 2/\varepsilon - 1)$ for which (12) holds with ε replaced by $\varepsilon/2$.*

Proof. The function $g(x) = x/(1+x)$ with domain $(0, +\infty)$ is strictly increasing and its range is $(0, 1)$. So, for every i, we can find $\mu'(A_i) < \mu(A_i)$ such that

$$\frac{\mu'(A_i)}{1+\mu'(A_i)} = (1-\varepsilon/2)\frac{\mu(A_i)}{1+\mu(A_i)} \tag{15}$$

$$< 1-\varepsilon/2 \ . \tag{16}$$

This choice implies that μ' is bounded, as desired, since starting with (16) we get

$$\frac{\mu'(A_i)}{1+\mu'(A_i)} < 1-\varepsilon/2 \implies \mu'(A_i) < 1+\mu'(A_i) - \frac{\varepsilon}{2}(1+\mu'(A_i)) \implies \mu'(A_i) < \frac{2}{\varepsilon} - 1 \ . \tag{17}$$

Now, starting with (14) and using that $(1-\varepsilon/2)^2 > 1-\varepsilon$ for any $\varepsilon > 0$ we get (18) below. Substituring (15) in (18) yields (19). Using that $0 < \mu' < \mu$ yields (20). Reorganizing the sum in (20) as in the derivation of (13) yields (21), i.e., the conclusion of the lemma.

$$\Pr[A_i] < (1-\varepsilon/2)^2 \cdot \frac{\mu(A_i)}{1+\mu(A_i)} \left(\sum_{U \in I_i \setminus \{A_i\}} \prod_{B \in U} \mu(u) \right)^{-1} \tag{18}$$

$$= (1-\varepsilon/2) \cdot \frac{\mu'(A_i)}{1+\mu'(A_i)} \cdot \left(\sum_{U \in I_i \setminus \{A_i\}} \prod_{B \in U} \mu(u) \right)^{-1} \tag{19}$$

$$< (1-\varepsilon/2) \cdot \frac{\mu'(A_i)}{1+\mu'(A_i)} \cdot \left(\sum_{U \in I_i \setminus \{A_i\}} \prod_{B \in U} \mu'(u) \right)^{-1} \tag{20}$$

$$= (1-\varepsilon/2) \cdot \mu'(A_i) \cdot \left(\sum_{U \in S_i} \prod_{B \in U} \mu'(B) \right)^{-1} \ . \tag{21}$$

◀

By Lemma 9, we know that (21) holds for all i. Since $\mu' > 0$ and $I_i \subseteq S_i$, this immediately implies that for all A_i,

$$\Pr[A_i] \leq (1-\varepsilon/2)\mu'(A_i) \cdot \left(\sum_{U \in I_i} \prod_{B \in U} \mu'(B) \right)^{-1} \ ,$$

which is precisely the condition of Theorem 4. Applying the theorem yields that the expected number of resamplings is $O\left(\varepsilon^{-1} n \log\left(\sum_{A_i \in \mathcal{A}} \mu'(A_i)\right)\right)$. But by (17)

$$\sum_{A_i \in \mathcal{A}} \mu'(A_i) \leq (1 + \max_i \mu'(A_i)) \sum_{A_i \in \mathcal{A}} \frac{\mu'(A_i)}{1+\mu'(A_i)} < (2/\varepsilon) Z(\mu') \ . \tag{22}$$

References

1 Noga Alon. A parallel algorithmic version of the Local Lemma. In *FOCS*, pages 586–593. IEEE Computer Society, 1991.

2 József Beck. An algorithmic approach to the Lovász Local Lemma. I. *Random Struct. Algorithms*, 2(4):343–366, 1991.

3 Rodrigo Bissacot, Roberto Fernández, Aldo Procacci, and Benedetto Scoppola. An improvement of the Lovász Local Lemma via cluster expansion. *Combinatorics, Probability & Computing*, 20(5):709–719, 2011.

4 Paul Erdös and Laszlo Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In A. Hajnal et al., editors, *Infinite and Finite Sets*, volume 11 of *Colloq. Math. Soc. Janos Bolyai*, pages 609–627. North-Holland, 1975.

5 Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the Lovász Local Lemma. In *FOCS*, pages 397–406. IEEE Computer Society, 2010.

6 Kashyap Babu Rao Kolipaka and Mario Szegedy. Moser and Tardos meet Lovász. In *STOC*, pages 235–244. ACM, 2011.

7 Michael Molloy and Bruce A. Reed. Further algorithmic aspects of the Local Lemma. In *STOC*, pages 524–529. ACM, 1998.

8 Robin A. Moser. A constructive proof of the Lovász Local Lemma. In *STOC*, pages 343–350. ACM, 2009.

9 Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász Local Lemma. *J. ACM*, 57(2), 2010.

10 Wesley Pegden. An improvement of the Moser-Tardos algorithmic local lemma. *CoRR*, abs/1102.2853, 2011.

11 Aravind Srinivasan. Improved algorithmic versions of the Lovász Local Lemma. In *SODA*, pages 611–620. SIAM, 2008.

Test Generation Using Symbolic Execution

Patrice Godefroid

Microsoft Research
pg@microsoft.com

Abstract

This paper presents a short introduction to automatic code-driven test generation using symbolic execution. It discusses some key technical challenges, solutions and milestones, but is not an exhaustive survey of this research area.

1998 ACM Subject Classification D.2.5 Testing and Debugging, D.2.4 Software/Program Verification

Keywords and phrases Testing, Symbolic Execution, Verification, Test Generation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.24

1 Automatic Code-Driven Test Generation

In this paper, we discuss the problem of *automatic code-driven test generation*:

> Given a program with a known set of input parameters, automatically generate a set of input values that will exercise as many program statements as possible.

Variants of this problem definition can be obtained using other code coverage criteria [48]. An optimal solution to this problem is theoretically impossible since this problem is undecidable in general (for infinite-state programs written in Turing-expressive programming languages). In practice, approximate solutions are sufficient.

Although automating test generation using program analysis is an old idea (e.g., [41]), practical tools have only started to emerge during the last few years. Indeed, the expensive sophisticated program-analysis techniques required to tackle the problem, such as symbolic execution engines and constraint solvers, have only become computationally affordable in recent years thanks to the increasing computational power available on modern computers. Moreover, this steady increase in computational power has in turn enabled recent progress in the engineering of more practical software analysis techniques. Specifically, this recent progress was enabled by new advances in dynamic test generation [29], which generalizes and is more powerful than static test generation, as explained later in this paper.

Automatic code-driven test generation differs from *model-based testing*. Given an abstract representation of the program, called *model*, model-based testing consists in generating tests to check the *conformance* of the program with respect to the model. In contrast, code-driven test generation does not use or require a model of the program under test. Instead, its goal is to generate tests that exercise as many program statements as possible, including assertions inserted in the code if any. Another fundamental difference is that models are usually written in abstract formal modeling languages which are, by definition, more amenable to precise analysis and test generation. In contrast, code-driven test generation has to deal with arbitrary software code and systems for which program analysis is bound to be imprecise, as discussed below.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 24–33

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

2 Symbolic Execution

Symbolic execution is a program analysis technique that was introduced in the 70s (e.g., see [41, 5, 14, 53, 39]). Symbolic execution means executing a program with symbolic rather than concrete values. Assignment statements are represented as functions of their (symbolic) arguments, while conditional statements are expressed as constraints on symbolic values. Symbolic execution can be used for many purposes, such as bug detection, program verification, debugging, maintenance, and fault localization [15].

One of the earliest proposals for using static analysis as a kind of symbolic program testing method was proposed by King almost 35 years ago [41]. The idea is to symbolically explore the tree of all computations the program exhibits when all possible value assignments to input parameters are considered. For each *control path* ρ, that is, a sequence of control locations of the program, a *path constraint* ϕ_ρ is constructed that characterizes the input assignments for which the program executes along ρ. All the paths can be enumerated by a search algorithm that explores all possible branches at conditional statements. The paths ρ for which ϕ_ρ is satisfiable are *feasible* and are the only ones that can be executed by the actual program. The solutions to ϕ_ρ characterize the inputs that drive the program through ρ. This characterization is exact provided symbolic execution has perfect precision. Assuming that the theorem prover used to check the satisfiability of all formulas ϕ_ρ is sound and complete, this use of static analysis amounts to a kind of symbolic testing.

A prototype of this system allowed the programmer to be presented with feasible paths and to experiment with assertions in order to force new and perhaps unexpected paths. King noticed that assumptions, now called preconditions, also formulated in the logic could be joined to the analysis forming, at least in principle, an automated theorem prover for Floyd/Hoare's verification method, including inductive invariants for programs that contain loops. Since then, this line of work has been developed further in various ways, leading to various strands of program verification approaches, such as *verification-condition generation* (e.g., [20, 37, 17, 3]), *symbolic model checking* [6] and *bounded model checking* [13].

Symbolic execution is also a key ingredient for *precise* automatic code-driven test generation. While program verification aims at proving the absence of program errors, test generation aims at generating concrete test inputs that can drive the program to execute specific program statements or paths.

Work on automatic code-driven test generation using symbolic execution can roughly be partitioned into two groups: *static* versus *dynamic* test generation.

3 Static Test Generation

Static test generation (e.g., [41]) consists of analyzing a program P statically, by using symbolic execution techniques to attempt to compute inputs to drive P along specific execution paths or branches, *without ever executing the program.*

Unfortunately, this approach is ineffective whenever the program contains statements involving constraints outside the scope of reasoning of the theorem prover, i.e., statements "that cannot be reasoned about symbolically". This limitation is illustrated by the following example [23]:

```
int obscure(int x, int y) {
  if (x == hash(y)) return -1;   // error
  return 0;                      // ok
}
```

Assume the constraint solver cannot "symbolically reason" about the function `hash` (perhaps because it is too complex or simply because its code is not available). This means that the constraint solver cannot generate two values for inputs `x` and `y` that are guaranteed to satisfy (or violate) the constraint `x == hash(y)`. In this case, static test generation cannot generate test inputs to drive the execution of the program `obscure` through either branch of the conditional statement: static test generation is *helpless* for a program like this. Note that, for test generation, it is not sufficient to know that the constraint `x == hash(y)` is satisfiable for *some* values of `x` and `y`, it is also necessary to generate *specific values* for `x` and `y` that satisfy or violate this constraint.

The practical implication of this simple observation is significant: static test generation is doomed to perform poorly whenever precise symbolic execution is not possible. Unfortunately, this is frequent in practice due to complex program statements (pointer manipulations, floating-point operations, etc.) and calls to operating-system and library functions that are hard or impossible to reason about symbolically with good enough precision.

4 Dynamic Test Generation

A second approach to test generation is *dynamic test generation* (e.g., [42, 49, 36]): it consists of executing the program P, typically starting with some random inputs, while performing symbolic execution *dynamically*, collecting symbolic constraints on inputs gathered from predicates in branch statements along the execution, and then using a constraint solver to infer variants of the previous inputs in order to steer the next execution of the program towards an alternative program branch. This process is repeated until a given final statement is reached or a specific program path is executed.

Directed Automated Random Testing [29], or DART for short, is a recent variant of dynamic test generation that blends it with model checking techniques with the goal of systematically executing *all* feasible program paths of a program while detecting various types of errors using run-time checking tools (like Purify, for instance). In DART, each new input vector attempts to force the execution of the program through *some* new path. By repeating this process, such a *directed search* attempts to force the program to sweep through all its feasible execution paths, in a style similar to *systematic testing* and *dynamic software model checking* [22].

In practice, a directed search typically cannot explore all the feasible paths of large programs in a reasonable amount of time. However, it usually does achieve better coverage than pure random testing and, hence, can find new program bugs.

A key observation [29] is that *imprecision in symbolic execution can be alleviated using concrete values and randomization*: whenever symbolic execution does not know how to generate a constraint for a program statement depending on some inputs, one can always simplify this constraint using the concrete values of those inputs.

Let us illustrate this important point with an example. Consider again the program `obscure` given above. Even though it is impossible to generate two values for inputs `x` and `y` such that the constraint `x == hash(y)` is satisfied (or violated), it is easy to generate, for a fixed value of `y`, a value of `x` that is equal to `hash(y)` since the latter can be observed and known at runtime. By picking randomly and then fixing the value of `y`, we first run the program, observe the concrete value c of `hash(y)` for that fixed value of `y` in that run; then, in the next run, we set the value of the other input `x` either to c or to another value, while leaving the value of `y` unchanged, in order to force the execution of the `then` or `else` branches, respectively, of the conditional statement in the function `obscure`. The DART

algorithm does all this automatically [29].

In summary, static test generation is unable to generate test inputs to control the execution of the program `obscure`, while dynamic test generation can *easily* drive the executions of that same program through all its feasible program paths. In realistic programs, imprecision in symbolic execution typically creeps in in many places, and dynamic test generation allows test generation to recover from that imprecision. Dynamic test generation can be viewed as extending static test generation with additional runtime information, and is therefore more general, precise, and powerful.

5 The Quest for Maximum Precision

Dynamic test generation is the most precise general form of code-driven test generation that is known today. It is more precise than static test generation and other forms of test generation such as random, taint-based and coverage-heuristic-based test generation. It is also the most sophisticated, requiring the use of automated theorem proving for solving path constraints. This machinery is more complex and heavy-weight, but may excerise more paths, find more bugs and generate fewer redundant tests covering the same path. Whether this maximum precision is worth the trouble depends on the application domain.

How much more precise is dynamic test generation compared to static test generation? In [24], it is shown exactly when the "concretization trick" used in the above `hash` example helps, or when it does not help. This is done formally by simulating the process of simplifying complex symbolic expressions using their runtime values using *uninterpreted functions*. Path constraints are then extended with uninterpreted function symbols representing imprecision during symbolic execution. For test generation, it is shown that those uninterpreted function symbols need to be *universally quantified*, unlike variables representing ordinary program inputs which are *existentially quantified*. In other words, this *higher-order* representation of path constraints forces test generation to be done from *validity proofs* of first-order logic formulas with uninterpreted functions, instead of *satisfiability proofs* of quantifier-free logic formulas (without uninterpreted functions) as usual.

The bottom-line is this: the key property of dynamic test generation that makes it more powerful than static test generation is *only* its ability to observe concrete values and to record those in path constraints. In contrast, the process of simplifying complex symbolic expressions using concrete runtime values can be accurately simulated statically using uninterpreted functions. However, those concrete values are necessary to effectively compute new input vectors, a fundamental requirement in test generation [24].

In principle, static test generation can be extended to concretize symbolic values whenever static symbolic execution becomes imprecise [40]. In practice, this is problematic and expensive because this approach not only requires to detect *all* sources of imprecision, but also requires one call to the constraint solver for each concretization to ensure that every synthesized concrete value satisfies prior symbolic constraints along the current program path. In contrast, dynamic test generation avoids these two limitations by leveraging a specific concrete execution as an automatic fall back for symbolic execution [29].

6 Whitebox Fuzzing (The Killer App)

Another significant recent milestone is the emergence of *whitebox fuzzing* [32] as the current main *"killer app"* for dynamic test generation, and arguably for automatic code-driven test generation in general.

Whitebox fuzzing extends dynamic test generation from unit testing to whole-program security testing. There are three main differences. First, inspired by so-called blackbox fuzzing [21], whitebox fuzzing performs dynamic test generation starting from one or several *well-formed* inputs, which is a heuristics to increase code coverage quickly and give the search a head-start. Second, again like blackbox fuzzing, the focus of whitebox fuzzing is to find *security vulnerabilities*, like buffer overflows, not to check functional correctness; finding such security vulnerabilities can be done fully automatically and does not require an application-specific test *oracle* or functional specification. Third, and more importantly, the main technical novelty of whitebox fuzzing is *scalability*: it extends the scope of dynamic test generation from (small) units to (large) whole programs. Whitebox fuzzing scales to large file parsers embedded in applications with millions of lines of code and execution traces with hundreds of millions of machine instructions.

Because whitebox fuzzing targets large applications, symbolic execution must scale to very long program executions, and is expensive. For instance, a single symbolic execution of Microsoft Excel with 45,000 input bytes executes nearly a billion x86 instructions. In this context, whitebox fuzzing uses a novel directed search algorithm, dubbed *generational search*, that maximizes the number of new input tests generated from each symbolic execution. Given a path constraint, *all* the constraints in that path are systematically negated one-by-one, placed in a conjunction with the prefix of the path constraint leading to it, and attempted to be solved by a constraint solver. This way, a single symbolic execution can generate thousands of new tests. (In contrast, a standard depth-first or breadth-first search would negate only the last or first constraint in each path constraint, and generate at most one new test per symbolic execution.)

Whitebox fuzzing was first implemented in the tool SAGE, short for *Scalable Automated Guided Execution* [32]. SAGE uses several optimizations that are crucial for dealing with huge execution traces. SAGE was also the first tool to perform dynamic symbolic execution at the x86 binary level. Working at the x86 binary level allows SAGE to be used on any program regardless of its source language or build process. It also ensures that *"what you fuzz is what you ship"* as compilers can perform source-code changes which may impact security.

Over the last few years, whitebox fuzzers have found many new security vulnerabilities (buffer overflows) in many Windows [32] and Linux [46] applications, including image processors, media players, file decoders, and document parsers.

Notably, SAGE found roughly one third of *all* the bugs discovered by file fuzzing during the development of Microsoft's Windows 7 [33], saving millions of dollars by avoiding expensive security patches for nearly a billion PCs worldwide. Because SAGE was typically run last, those bugs were missed by everything else, including static program analysis and blackbox fuzzing.

Since 2008, SAGE has been running *non-stop* on an average of 100+ machines, automatically fuzzing hundreds of applications in Microsoft security testing labs. This is over 400 machine-years and the *largest computational usage ever for any Satisfiability-Modulo-Theories (SMT) solver*, according to the authors of the Z3 SMT solver [16], with over three billion constraints processed to date.

7 Other Related Work

Over the last several years, other tools implementing dynamic test generation have been developed for various programming languages, properties and application domains. Examples

of such tools are DART [29], EGT [9], CUTE [58], EXE [10], Catchconv [47], PEX [60], KLEE [8], CREST [7], BitBlaze [59], Splat [45], Apollo [2], and YOGI [35], to name some. Dynamic test generation has become so popular that it is also sometimes casually referred to as "execution-generated tests" [9], "concolic testing" [58], or simply "dynamic symbolic execution" [60].

All the above tools differ by how they perform symbolic execution (for languages such as C, Java, x86, .NET, etc.), by the type of constraints they generate (for theories such as linear arithmetic, bit-vectors, arrays, uninterpreted functions, etc.), and by the type of constraint solvers they use (such as lp_solve, CVClite, STP, Disolver, Yikes, Z3, etc.). Indeed, like in traditional static program analysis and abstract interpretation, these important parameters depend in practice on which type of program is to be tested, on how the program interfaces with its environment, and on the property of interest. Moreover, various cost/precision tradeoffs are also possible, as usual in program analysis.

When building tools like these, there are many other challenges, such as: how to recover from imprecision in symbolic execution [29, 24], how to check efficiently many properties together [10, 31], how to leverage grammars (when available) for complex input formats [44, 27], how to deal with path explosion [23, 1, 4, 45, 35], how to precisely reason about pointers [58, 10, 18], how to deal with inputs of varying sizes [61], how to deal with floating-point instructions [28], how to deal with input-dependent loops [55, 34], which heuristics to prioritize the search in the program's search space [10, 32, 7], how to re-use previous analysis results across code changes [51, 30, 52], how to leverage reachability facts inferred by static program analysis [35], etc. Other recent work has also explored how to target other application areas, such as concurrent programs [57], database applications [19], web applications [2, 54], or device drivers [35, 43].

More broadly, many other papers discussing test generation and program verification have been published over the last 30+ years. It would be virtually impossible to survey them all. We only highlighted here some key technical problems and recent milestones. We encourage the reader to consult other recent surveys, such as [11, 56, 50, 25], which present different, yet also partial, points of view.

8 Conclusion

Automatic code-driven test generation aims at proving existential properties of programs: does there exist a test input that can exercise a specific program branch or statement, or follow a specific program path, or trigger a bug? Test generation dualizes traditional program verification and static program analysis aimed at proving universal properties which holds for all program paths, such as "there are no bugs of type X in this program".

Symbolic reasoning about large programs is bound to be imprecise. If perfect bit-precise symbolic reasoning was possible, static program analysis would detect standard programming errors without reporting false alarms. How to deal with this imprecision is a fundamental problem in program analysis. Traditional static program verification builds "may" over-approximations of the program behaviors in order to prove correctness, but at the cost of reporting false alarms. Dually, automatic test generation requires "must" under-approximations in order to drive program executions and find bugs without reporting false alarms, but at the cost of possibly missing bugs.

Test generation is only one way of proving existential reachability properties of programs, where specific concrete input values are generated to exercise specific program paths. More generally, such properties can be proved using so-called *must abstractions* of programs [26],

without necessarily generating concrete tests. A must abstraction is defined as a program abstraction that preserves existential reachability properties of the program. Sound path constraints are particular cases of must abstractions [35]. Must abstractions can also be built backwards from error states using static program analysis [12, 38]. This approach can detect program locations and states provably leading to error states (no false alarms), but may fail to prove reachability of those error states back from whole-program initial states, and hence may miss bugs or report unreachable error states.

Most tools mentioned in the previous section are research prototypes, aimed at exploring new ideas, but they are not used on a daily basis by ordinary software developers and testers. Finding other "killer apps" for these techniques, beyond whitebox fuzzing of file and packet parsers, is *critical* in order to sustain progress in this research area.

References

1 S. Anand, P. Godefroid, and N. Tillmann. Demand-Driven Compositional Symbolic Execution. In *Proceedings of TACAS'2008 (14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems)*, volume 4963 of *Lecture Notes in Computer Science*, pages 367–381, Budapest, April 2008. Springer-Verlag.

2 S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. M. Paradkar, and M. D. Ernst. Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit-State Model Checking. *IEEE Trans. Software Eng.*, 36(4):474–494, 2010.

3 M. Barnett, B. E. Chang, R. DeLine, B. Jacobs, and K. R. M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In *Proceedings of FMCO'2005 (4th International Symposium on Formal Methods for Components and Objects)*, volume 4111 of *Lecture Notes in Computer Science*, pages 364–387. Springer-Verlag, September 2006.

4 P. Boonstoppel, C. Cadar, and D. Engler. RWset: Attacking path explosion in constraint-based test generation. In *TACAS'08*, April 2008.

5 R. S. Boyer, B. Elspas, and K. N. Levitt. SELECT – a formal system for testing and debugging programs by symbolic execution. *SIGPLAN Not.*, 10:234–245, 1975.

6 J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of LICS'1990 (5th Symposium on Logic in Computer Science)*, pages 428–439, Philadelphia, June 1990.

7 J. Burnim and K. Sen. Heuristics for scalable dynamic test generation. In *ASE'08*, 2008.

8 C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI'08*, Dec 2008.

9 C. Cadar and D. Engler. Execution Generated Test Cases: How to Make Systems Code Crash Itself. In *Proceedings of SPIN'2005 (12th International SPIN Workshop on Model Checking of Software)*, volume 3639 of *Lecture Notes in Computer Science*, San Francisco, August 2005. Springer-Verlag.

10 C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. EXE: Automatically Generating Inputs of Death. In *ACM CCS*, 2006.

11 C. Cadar, P. Godefroid, S. Khurshid, C.S. Pasareanu, K. Sen, N.Tillmann, and W. Visser. Symbolic Execution for Software Testing in Practice – Preliminary Assessment. In *ICSE'2011*, Honolulu, May 2011.

12 S. Chandra, S. J. Fink, and M. Sridharan. Snugglebug: A Powerful Approach to Weakest Preconditions. In *Proceedings of PLDI'2009 (ACM SIGPLAN 2009 Conference on Programming Language Design and Implementation)*, Dublin, June 2009.

13 E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

14 L. A. Clarke. A program testing system. In *Proc. of the 1976 annual conference*, pages 488–491, 1976.

15 L. A. Clarke and D. J. Richardson. Applications of symbolic evaluation. *Journal of Systems and Software*, 5(1):15–35, 1985.

16 L. de Moura and N. Bjorner. Z3: An Efficient SMT Solver. In *Proceedings of TACAS'2008 (14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Budapest, April 2008. Springer-Verlag.

17 E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18:453–457, 1975.

18 B. Elkarablieh, P. Godefroid, and M.Y. Levin. Precise Pointer Reasoning for Dynamic Test Generation. In *Proceedings of ISSTA'09 (ACM SIGSOFT International Symposium on Software Testing and Analysis)*, pages 129–139, Chicago, July 2009.

19 M. Emmi, R. Majumdar, and K. Sen. Dynamic Test Input Generation for Database Applications. In *Proceedings of ISSTA'2007 (International Symposium on Software Testing and Analysis*, pages 151–162, 2007.

20 R. Floyd. Assigning Meaning to Programs. In *Mathematical Aspects of Computer Science*, pages 19–32. XIX American Mathematical Society, 1967.

21 J. E. Forrester and B. P. Miller. An Empirical Study of the Robustness of Windows NT Applications Using Random Testing. In *Proceedings of the 4th USENIX Windows System Symposium*, Seattle, August 2000.

22 P. Godefroid. Model Checking for Programming Languages using VeriSoft. In *Proceedings of POPL'97 (24th ACM Symposium on Principles of Programming Languages)*, pages 174–186, Paris, January 1997.

23 P. Godefroid. Compositional Dynamic Test Generation. In *Proceedings of POPL'2007 (34th ACM Symposium on Principles of Programming Languages)*, pages 47–54, Nice, January 2007.

24 P. Godefroid. Higher-Order Test Generation. In *Proceedings of PLDI'2011 (ACM SIGPLAN 2011 Conference on Programming Language Design and Implementation)*, pages 258–269, San Jose, June 2011.

25 P. Godefroid, P. de Halleux, M. Y. Levin, A. V. Nori, S. K. Rajamani, W. Schulte, and N. Tillmann. Automating Software Testing Using Program Analysis. *IEEE Software*, 25(5):30–37, September/October 2008.

26 P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based Model Checking using Modal Transition Systems. In *Proceedings of CONCUR'2001 (12th International Conference on Concurrency Theory)*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440, Aalborg, August 2001. Springer-Verlag.

27 P. Godefroid, A. Kiezun, and M. Y. Levin. Grammar-based Whitebox Fuzzing. In *Proceedings of PLDI'2008 (ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation)*, pages 206–215, Tucson, June 2008.

28 P. Godefroid and J. Kinder. Proving Memory Safety of Floating-Point Computations by Combining Static and Dynamic Program Analysis. In *Proceedings of ISSTA'2010 (ACM SIGSOFT International Symposium on Software Testing and Analysis)*, pages 1–11, Trento, July 2010.

29 P. Godefroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In *Proceedings of PLDI'2005 (ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation)*, pages 213–223, Chicago, June 2005.

30 P. Godefroid, S. K. Lahiri, and C. Rubio-Gonzalez. Statically Validating Must Summaries for Incremental Compositional Dynamic Test Generation. In *Proceedings of SAS'2011*

(18th International Static Analysis Symposium), volume 6887 of *Lecture Notes in Computer Science*, pages 112–128, Venice, September 2011. Springer-Verlag.

31 P. Godefroid, M.Y. Levin, and D. Molnar. Active Property Checking. In *Proceedings of EMSOFT'2008 (8th Annual ACM & IEEE Conference on Embedded Software)*, pages 207–216, Atlanta, October 2008. ACM Press.

32 P. Godefroid, M.Y. Levin, and D. Molnar. Automated Whitebox Fuzz Testing. In *Proceedings of NDSS'2008 (Network and Distributed Systems Security)*, pages 151–166, San Diego, February 2008.

33 P. Godefroid, M.Y. Levin, and D. Molnar. SAGE: Whitebox Fuzzing for Security Testing. *Communications of the ACM*, 55(3):40–44, March 2012.

34 P. Godefroid and D. Luchaup. Automatic Partial Loop Summarization in Dynamic Test Generation. In *Proceedings of ISSTA'2011 (ACM SIGSOFT International Symposium on Software Testing and Analysis)*, pages 23–33, Toronto, July 2011.

35 P. Godefroid, A.V. Nori, S.K. Rajamani, and S.D. Tetali. Compositional May-Must Program Analysis: Unleashing The Power of Alternation. In *Proceedings of POPL'2010 (37th ACM Symposium on Principles of Programming Languages)*, pages 43–55, Madrid, January 2010.

36 N. Gupta, A. P. Mathur, and M. L. Soffa. Generating Test Data for Branch Coverage. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering*, pages 219–227, September 2000.

37 C. A. R. Hoare. An Axiomatic Approach to Computer Programming. *Communications of the ACM*, 12(10):576–580, 1969.

38 J. Hoenicke, K. R. M. Leino, A. Podelski, M. Schaf, and Th. Wies. It's doomed; we can prove it. In *Proceedings of 2009 World Congress on Formal Methods*, 2009.

39 W.E. Howden. Symbolic testing and the DISSECT symbolic evaluation system. *IEEE Transactions on Software Engineering*, 3(4):266–278, 1977.

40 S. Khurshid, C. Pasareanu, and W. Visser. Generalized Symbolic Execution for Model Checking and Testing. In *TACAS'03*, April 2003.

41 J. C. King. Symbolic Execution and Program Testing. *Journal of the ACM*, 19(7):385–394, 1976.

42 B. Korel. A Dynamic Approach of Test Data Generation. In *IEEE Conference on Software Maintenance*, pages 311–317, San Diego, November 1990.

43 V. Kuznetsov, V. Chipounov, and G. Candea. Testing closed-source binary device drivers with DDT. In *USENIX ATC'10*, June 2010.

44 R. Majumdar and R. Xu. Directed Test Generation using Symbolic Grammars. In *ASE*, 2007.

45 R. Majumdar and R. Xu. Reducing test inputs using information partitions. In *CAV'09*, pages 555–569, 2009.

46 D. Molnar, X. C. Li, and D. Wagner. Dynamic test generation to find integer bugs in x86 binary linux programs. In *Proc. of the 18th Usenix Security Symposium*, Aug 2009.

47 D. Molnar and D. Wagner. Catchconv: Symbolic execution and run-time type inference for integer conversion errors, 2007. UC Berkeley EECS, 2007-23.

48 G. J. Myers. *The Art of Software Testing*. Wiley, 1979.

49 A. J. Offutt, Z. Jin, and J. Pan. The Dynamic Domain Reduction Procedure for Test Data Generation. *Software Practice and Experience*, 29(2):167–193, 1997.

50 C. S. Pasareanu and W. Visser. A survey of new trends in symbolic execution for software testing and analysis. *STTT*, 11(4):339–353, 2009.

51 S. Person, M. B. Dwyer, S. G. Elbaum, and C. S. Pasareanu. Differential symbolic execution. In *SIGSOFT FSE*, pages 226–237, 2008.

52 S. Person, G. Yang, N. Rungta, and S. Khurshid. Directed Incremental Symbolic Execution. In *PLDI'2011*, pages 504–515, San Jose, June 2011.

53 C.V. Ramamoorthy, S.-B.F. Ho, and W.T. Chen. On the automated generation of program test data. *IEEE Trans. on Software Engineering*, 2(4):293–300, 1976.

54 P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. A Symbolic Execution Framework for JavaScript. In *IEEE Symposium on Security and Privacy*, pages 513–528, 2010.

55 P. Saxena, P. Poosankam, S. McCamant, and D. Song. Loop-Extended Symbolic Execution on Binary Programs. In *ISSTA'2009*, pages 225–236, Chicago, July 2009.

56 E. J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *IEEE Symposium on Security and Privacy*, May 2010.

57 K. Sen and G. Agha. CUTE and jCUTE : Concolic unit testing and explicit path model-checking tools. In *CAV'06*, 2006.

58 K. Sen, D. Marinov, and G. Agha. CUTE: A Concolic Unit Testing Engine for C. In *Proceedings of FSE'2005 (13th International Symposium on the Foundations of Software Engineering)*, Lisbon, September 2005.

59 D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. BitBlaze: A New Approach to Computer Security via Binary Analysis. In *ICISS'2008*, December 2008.

60 N. Tillmann and J. de Halleux. Pex - White Box Test Generation for .NET. In *Proceedings of TAP'2008 (2nd International Conference on Tests and Proofs)*, volume 4966 of *Lecture Notes in Computer Science*, pages 134–153. Springer-Verlag, April 2008.

61 R. Xu, , P. Godefroid, and R. Majumdar. Testing for Buffer Overflows with Length Abstraction. In *Proceedings of ISSTA'08 (ACM SIGSOFT International Symposium on Software Testing and Analysis)*, pages 27–38, Seattle, July 2008.

Automated Reasoning and Natural Proofs for Programs Manipulating Data Structures

P. Madhusudan

University of Illinois at Urbana-Champaign, USA
and visiting Microsoft Research, Bangalore, INDIA
madhu@illinois.edu

Abstract

We consider the problem of automatically verifying programs that manipulate a dynamic heap, maintaining complex and multiple data-structures, given modular pre-post conditions and loop invariants. We discuss specification logics for heaps, and discuss two classes of automatic procedures for reasoning with these logics. The first identifies fragments of logics that admit completely decidable reasoning. The second is a new approach called the *natural proof method* that builds proof procedures for very expressive logics that are automatic and sound (but incomplete), and that embody natural proof tactics learnt from manual verification.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic

Keywords and phrases logic, heap structures, data structures, program verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.34

Summary

One of the most promising paradigms of software verification is *automated deductive verification*, which combines user written modular annotations for specifications as well as invariants (pre/post conditions, loop invariants, assertions, ghost code, etc.) and automatic theorem proving of the resulting verification conditions. This paradigm is extremely powerful as it appears to be a rich enough paradigm using which any reliable software can be built (unlike completely automatic approaches) and because the user needs to specify only modular and loop annotations, leaving reasoning entirely to automatic techniques. Several success stories of large software verification projects attest to the power of this paradigm (the Verve OS project [8], the Microsoft hypervisor verification project using VCC [1], and a recent verified-for-security OS+browser for mobile applications [5], to name a few).

Verification conditions do not, however, always fall into decidable theories. In particular, the verification of properties of the *dynamically modified heap* is a big challenge for logical methods. The dynamically manipulated heap poses several challenges, as typical correctness properties of heaps require complex combinations of structure (e.g., p points to a tree structure, or to a doubly-linked list, or to an almost balanced tree, with respect to certain pointer-fields), data (the integers stored in data-fields of the tree respect the binary search tree property, or the data stored in a tree is a max-heap), and separation (the procedure modifies one list and not the other and leaves the two lists disjoint at exit, etc.). The fact that the dynamic heap contains an unbounded number of locations means that expressing the above properties requires *quantification* in some form, which immediately precludes the use of most decidable theories currently handled by SMT solvers.

We will discuss two logics that have emerged in this regime— classical logic augmented with ghost-code and separation logic [6]. We will then discuss two thrusts in automatically verifying the resulting verification conditions— decidable logics and natural proofs.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 34–35

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

Decidable logics: We discuss several decidable logics that restrict classical logics with quantification so that they are amenable to automated reasoning. In particular, we will discuss the class of STRAND logics [2, 3] that admit decision procedures for data-structures combining tree-interpretable structure and data, with restriction of relations to be *elastic*. We will explore the boundary between decidability and undecidability in this domain.

Decidable logics that are expressive enough to handle common data-structures and their correctness quickly become awkward, and perhaps more importantly, their decision procedures get incredibly complex, failing to mimic the simple proofs that are sufficient to prove many correct programs correct. This leads us to the quest for simple proofs.

Natural proofs: Natural proofs [4, 7] aim at discovering *simple* proofs— proofs that mimic human reasoning of programs, which often rely on *induction* on the shape of the data-structure, and combine unfolding recursive definitions on data-structures followed by unification and simple quantifier-free reasoning over specific theories. Natural proofs exploit a *fixed* set of proof tactics, keeping the expressiveness of powerful logics, retaining the automated nature of proving validity, but giving up on completeness (giving up decidability, retaining soundness).

We discuss the logic DRYAD, a dialect of separation logic, with no explicit (classical) quantification but with recursive definitions to express second-order properties. We show that DRYAD is both powerful in terms of expressiveness, and closed under the strongest-post with respect to bounded code segments. We also show that DRYAD can be systematically converted to classical logic using the theory of sets, and develop a natural proof mechanism for classical logics with recursion and sets that implements a sound but incomplete reduction to decidable theories that can be handled by an SMT solver.

We show, using a large class of correct programs manipulating lists, trees, cyclic lists, and doubly linked lists as well as multiple data-structures of these kinds, that the natural proof mechanism often succeeds in proving programs automatically. These programs are drawn from a range of sources, from textbook data-structure routines (binary search trees, red-black trees, etc.) to routines from Glib low-level C-routines used in GTK+/Gnome to routines implementing file-systems, a routine from the Schorr-Waite garbage collection algorithm, to several programs from a recent secure framework developed for mobile applications [5].

References

1 Ernie Cohen, Markus Dahlweid, Mark A. Hillebrand, Dirk Leinenbach, Michal Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. VCC: A practical system for verifying concurrent C. In *TPHOLs'09*, volume 5674 of *LNCS*, pages 23–42. Springer, 2009.

2 P. Madhusudan, Gennaro Parlato, and Xiaokang Qiu. Decidable logics combining heap structures and data. In *POPL'11*, pages 611–622. ACM, 2011.

3 P. Madhusudan and Xiaokang Qiu. Efficient decision procedures for heaps using STRAND. In *SAS'11*, volume 6887 of *LNCS*, pages 43–59. Springer, 2011.

4 P. Madhusudan, Xiaokang Qiu, and Andrei Stefanescu. Recursive proofs for inductive tree data-structures. In *POPL'12*, pages 123–136. ACM, 2012.

5 Haohui Mai, Edgar Pek, Hui Xue, P. Madhusudan, and Samuel King. Building a secure foundation for mobile apps. In *ASPLOS'13*. to appear, 2013.

6 Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *CSL'01*, volume 2142 of *LNCS*, pages 1–19. Springer, 2001.

7 Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and P. Madhusudan. Natural proofs for structure, data, and separation. Unpublished manuscript, 2012.

8 Jean Yang and Chris Hawblitzel. Safe to the last instruction: automated verification of a type-safe operating system. In *PLDI'10*, pages 99–110. ACM, 2010.

Certifying polynomials for $\mathrm{AC}^0[\oplus]$ circuits, with applications

Swastik Kopparty[1] and Srikanth Srinivasan[2]

1 **Rutgers University.** swastik.kopparty@rutgers.edu
2 **DIMACS, Rutgers University.** srikanth@dimacs.rutgers.edu

Abstract

In this paper, we introduce and develop the method of certifying polynomials for proving $\mathrm{AC}^0[\oplus]$ circuit lower bounds.

We use this method to show that Approximate Majority cannot be computed by $\mathrm{AC}^0[\oplus]$ circuits of size $n^{1+o(1)}$. This implies a separation between the power of $\mathrm{AC}^0[\oplus]$ circuits of near-linear size and uniform $\mathrm{AC}^0[\oplus]$ (and even AC^0) circuits of polynomial size. This also implies a separation between randomized $\mathrm{AC}^0[\oplus]$ circuits of linear size and deterministic $\mathrm{AC}^0[\oplus]$ circuits of near-linear size.

Our proof using certifying polynomials extends the deterministic restrictions technique of Chaudhuri and Radhakrishnan, who showed that Approximate Majority cannot be computed by AC^0 circuits of size $n^{1+o(1)}$. At the technical level, we show that for every $\mathrm{AC}^0[\oplus]$ circuit C of near-linear size, there is a low degree variety V over $\mathbb{F}_2$ such that the restriction of C to V is constant.

We also prove other results exploring various aspects of the power of certifying polynomials. In the process, we show an essentially optimal lower bound of $\Omega\left(\log^{\Theta(d)} s \cdot \log \frac{1}{\epsilon}\right)$ on the degree of ϵ-approximating polynomials for $\mathrm{AC}^0[\oplus]$ circuits of size s.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases Constant-depth Boolean circuits, Polynomials over finite fields, Size hierarchies

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.36

1 Introduction

In this paper, we introduce and develop the method of certifying polynomials for proving circuit lower bounds. We begin by describing the motivation for the main new circuit lower bound that we show, after which we will elaborate on the the method itself, and finally we describe some other results exploring the power and limitations of this method.

1.1 The Size Hierarchy Problem for $\mathrm{AC}^0[\oplus]$

Our main result fits in the general theme of studying the relative power of constant depth circuit classes. We show a near-tight circuit lower-bound for computing Approximate Majority with AND, OR, PARITY and NOT gates. This is a first step in the direction of a uniform size-hierarchy theorem for such circuits, which is a basic open question about this well-studied class of circuits.

We first fix some notation and conventions regarding circuits for the rest of this paper. AC^0 denotes the class of bounded depth circuits with unbounded fan-in AND, OR and NOT

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 36–47

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

gates. $\mathrm{AC}^0[\oplus]$ denotes the class of bounded depth circuits with unbounded fan-in AND, OR, PARITY and NOT gates. We measure the size of a circuit by the number of gates. We use n to denote the number of input bits to a circuit.

There is a well-developed theory giving superpolynomial and even subexponential lower bounds for AC^0 and $\mathrm{AC}^0[\oplus]$ circuits [6, 1, 15, 7, 9, 12]. Our focus in this paper is on complexity theory within these classes.

An influential paper of Ragde and Wigderson [8] asked if uniform AC^0 circuits of linear size are strictly weaker as uniform AC^0 circuits of polynomial size. This was answered by Chaudhuri and Radhakrishnan [5], who showed that Approximate Majority functions do not have AC^0 circuits of near-linear size $O(n^{1+\epsilon_d})$ (where $\epsilon_d > 0$). An Approximate Majority function is any function which maps strings of Hamming weight $< n/4$ to 0 and strings of Hamming weight $> 3n/4$ to 1. Such functions were first considered in the context of AC^0 for the purpose of error-reduction for AC^0 circuits. Ajtai and Ben-Or [2] showed that Approximate Majority can be computed by polynomial-size AC^0 circuits, and later results of Ajtai [1] and Viola [14] showed that this can even be done by uniform polynomial-size AC^0 circuits of depth 3 and above (In fact these circuits can be made to have depth d and size $O(n^{1+\epsilon_d})$, where $\epsilon_d \to 0$ as $d \to \infty$ [5]). This combined with the lower-bound of [5] showed the conjectured separation of Ragde and Wigderson.

The method of proof of [5] is especially interesting to us, and we will discuss their method and our extension of it in the next subsection.

A beautiful recent result of Rossman [11] showed a size-hierarchy for AC^0: for every integer $k > 0$, uniform AC^0 circuits of size $O(n^k)$ are more powerful than non-uniform AC^0 circuits of size $O(n^{k/4})$. A striking follow-up result of Amano [3] in fact shows that depth-2 size $O(n^k)$ uniform AC^0 circuits can be more powerful than size $O(n^{k-\epsilon})$ AC^0 circuits for arbitrary $\epsilon > 0$.

In this work we study the analogous questions for uniform $\mathrm{AC}^0[\oplus]$. Our main result is that Approximate Majority cannot be computed by $\mathrm{AC}^0[\oplus]$ circuits of near-linear size. In particular this means that polynomial size uniform $\mathrm{AC}^0[\oplus]$ circuits (and even polynomial size uniform AC^0 circuits) can be more powerful than near-linear size $\mathrm{AC}^0[\oplus]$ circuits. Thus we make a first step towards a size-hierarchy theorem for $\mathrm{AC}^0[\oplus]$ circuits, analogous to the result of Chaudhuri and Radhakrishnan for AC^0. Our result also shows that randomized $\mathrm{AC}^0[\oplus]$ circuits of linear size can be more powerful than deterministic $\mathrm{AC}^0[\oplus]$ circuits of near-linear size.

Showing the full size-hierarchy for uniform $\mathrm{AC}^0[\oplus]$ is still open and would be very interesting. Even the question of whether there exists a function that has uniform $\mathrm{AC}^0[\oplus]$ circuits of size $n^{\log n}$ but no polynomial-sized $\mathrm{AC}^0[\oplus]$ circuits (of possibly larger, but constant, depth) remains unanswered.

1.2 Certifying Polynomials for $\mathrm{AC}^0[\oplus]$

The main component of the [5] lower bound for Approximate Majority is a structure theorem for AC^0 circuits of near-linear size. It states that for every AC^0 circuit C of near-linear size, there is a collection of $o(n)$ variables and a fixing of them that simplifies the circuit C to a constant. Equivalently, there is a large axis-parallel subcube of $\{0,1\}^n$ on which C restricts to a constant. This structure theorem immediately implies the lower bound on Approximate Majority.

The proof of this structure theorem is by "deterministic restrictions". Going through the circuit in a bottom up fashion, one first finds a fixing of a small number of variables that simplifies the circuit into one where all the gates have small fan-in. The basic observation

is that if one considers the gates at height 1 that have large fan-in, then we can set a large number of them to constants by setting a few input variables; continuing in this way, we eventually remove all large fan-in gates of height 1 (there can't be too many of them, since C is of near-linear size), setting only a few variables in doing so. We then move on to higher levels and repeat the process, which now becomes feasible since setting gates of small fan-in to a constant reduces to setting only a few variables to constants. Once all the gates have small fan-in, the entire circuit is a function of only a few variables and hence, there is a fixing of small number of the remaining variables so that the circuit simplifies to a constant.

The main component of our lower bound is an analogous structure theorem for $\mathrm{AC}^0[\oplus]$. Clearly, the structure theorem for AC^0 is false for even a single parity gate and hence for $\mathrm{AC}^0[\oplus]$. However, here we can show that for for any $\mathrm{AC}^0[\oplus]$ circuit C of near-linear size, there is a polynomial of degree $o(n)$ such that C restricts to a constant on the zero-set of that polynomial. We call such a polynomial a *certifying polynomial* for the circuit C. The proof of this structure theorem again proceeds in a bottom up fashion, but this time finds fixings of systems of low-degree polynomials in order to simplify the circuit to one where all the AND and OR gates have small fan-in. Again, once all the AND and OR gates have small fan-in, it is easy to see that the circuit just computes a low-degree polynomial, and thus fixing this low-degree polynomial simplifies the circuit to a constant.

Given this structure theorem, it remains to see that no Approximate Majority function has this structure. This turns out to be a consequence of the general fact that a nonzero polynomial of degree d cannot vanish at every point of a Hamming ball of radius d (this follows from the fact that Hamming balls are interpolating sets for polynomials). In fact, it is even true that polynomials of degree $o(d)$ cannot vanish on all but an exponentially small fraction of a Hamming ball of radius d (this is a consequence of the p-biased version of the standard bound on the number of zeroes of a nonzero polynomial). We conclude that an Approximate Majority function cannot be constant on the zero set of a nonzero polynomial of degree $< n/4$. Combined with the structure theorem, this completes the proof of the lower bound for the $\mathrm{AC}^0[\oplus]$ complexity of Approximate Majority.

Having proved the lower bound, we then take a step back to re-examine the technique of proving lower bounds via certifying polynomials. On the face of it, it seems like this method is somewhat distinct from the Razborov-Smolensky method [9, 13] used to prove lower bounds for general $\mathrm{AC}^0[\oplus]$ circuits, which uses *polynomial approximations* to circuits. The Razborov-Smolensky method gives global, approximate structure: it shows that for any $\mathrm{AC}^0[\oplus]$ circuit C of size M, there is a polynomial of degree $\mathrm{poly}(\log(M))$ which agrees with C on most points of $\{0,1\}^n$. Our structure theorem, which only applies to circuits of near-linear size, gives local, exact structure: we get a perfect description of the values taken by an $\mathrm{AC}^0[\oplus]$ circuit on a small but structured subset of $\{0,1\}^n$.

As it turns out, however, the framework of certifying polynomials is quite robust: we demonstrate a connection between polynomial approximations and certifying polynomials for circuits. We then use this connection along with Razborov's approximating polynomials to construct certifying polynomials for general $\mathrm{AC}^0[\oplus]$ circuits. These polynomials have degree much larger than that obtained in our structure theorem, but nevertheless, their degree is small enough to be able to recover the exponential lower bound obtained by Razborov [9] for $\mathrm{AC}^0[\oplus]$ circuits computing the Majority function. We stress that most of the ideas of this lower bound proof are already present in [9, 13], and the main aim of this exercise is to show that the use of certifying polynomials is a unified framework that "explains" all previous lower bound approaches for $\mathrm{AC}^0[\oplus]$. In the course of the above proof, we also construct improved approximations to $\mathrm{AC}^0[\oplus]$ circuits in the small error regime; to the best

of our knowledge, such approximations were not known before, and may be of independent interest.

Finally, we exploit the connection between certifying polynomials and polynomial approximations in the reverse direction to prove limits on the power of polynomial approximations. We show that the low-error approximations we construct for $\mathrm{AC}^0[\oplus]$ are close to the best possible for all depths $d \geq 3$. Once again, this demonstrates the flexibility of the certifying polynomials framework.

2 Results

We begin by formally defining certifying polynomials. Throughout the paper, we identify $\{0,1\}$ with $\mathbb{F}_2$.

▶ **Definition 1** (Certifying polynomial). A polynomial $P(X_1, \ldots, X_n) \in \mathbb{F}_2[X_1, \ldots, X_n]$ is a certifying polynomial for a function $f : \{0,1\}^n \to \{0,1\}$ if:

- the set $S = \{x \in \mathbb{F}_2^n \mid P(x) = 0\}$ is nonempty,
- f is constant on S.

We now define Approximate Majority.

▶ **Definition 2** (Approximate Majority). An $(a, n-a)$ Approximate Majority is a boolean function $f : \{0,1\}^n \to \{0,1\}$ such that:

- $f(x) = 0$ for every x of Hamming weight at most a.
- $f(x) = 1$ for every x of Hamming weight at least $n - a$.

If we omit the $(a, n-a)$, we assume $a = n/4$.

Ajtai and Ben-Or [2] showed that for $a \leq n/2 - n/(\log n)^{O(1)}$, there exists an $(a, n-a)$ Approximate Majority computable in AC^0. We will use a uniform and more general version of this result, due to Ajtai [1].

▶ **Theorem 3** (Ajtai [1]). *For any $n \in \mathbb{N}$, $\delta \in (0, 1/2)$ and depth $d \geq 3$, there exist $((1/2-\delta)n, (1/2+\delta)n)$ Approximate Majorities computable by uniform AC^0 circuits of size $2^{(1/\delta)^{O(1/d)}} \cdot n^{O(1)}$ and depth d.*

Our main result is:

▶ **Theorem 4.** *For every constant $d \in \mathbb{N}$, there is an $\epsilon_d > 0$ such that any depth d $\mathrm{AC}^0[\oplus]$ circuit that computes an Approximate Majority must have size $\Omega(n^{1+\epsilon_d})$.*

By Theorem 3, this implies that uniform AC^0 circuits of polynomial size are more powerful than linear-sized non-uniform $\mathrm{AC}^0[\oplus]$ circuits.

The proof of Theorem 4 yields $\epsilon_d = 1/2^{d+1}$. This is marginally better than the lower bound of $\Omega(n^{1+1/4^d})$ obtained by Chaudhuri and Radhakrishnan [5] for the case of AC^0 (the improvement is due to a slightly different deterministic restriction method: whereas [5] try to remove both high fan-in and high fan-out gates from the circuit, we handle only the high fan-in gates). Also, as already showed by [5], this lower bound cannot be substantially improved since Approximate Majorities can be computed by AC^0 circuits of depth d and size $n^{1+1/2^{\Omega(d)}}$.

The proof of Theorem 4 follows from two lemmas. The first states that every function with a near-linear $\mathrm{AC}^0[\oplus]$ circuit has a certifying polynomial of low degree, The next states that an Approximate Majority cannot have this property. We now state these lemmas formally (the proofs appear in Section 3).

▶ **Lemma 5** (Linear-size $\mathrm{AC}^0[\oplus]$ circuits have low degree certifying polynomials)**.** *For every constant $d \in \mathbb{N}$, there is an $\epsilon_d > 0$ such that for every depth-d $\mathrm{AC}^0[\oplus]$ circuit C of size $s \leq n^{1+\epsilon_d}$, C has a certifying polynomial of degree $o(n)$.*

▶ **Lemma 6** (Approximate Majority does not have any low degree certifying polynomials)**.** *For every $(a, n-a)$ Approximate Majority f, there do not exist any certifying polynomials for f of degree $\leq a$.*

Next we state our results on certifying polynomials for general $\mathrm{AC}^0[\oplus]$ circuits. This result should be contrasted with the fact that every function has a certifying polynomial of degree at most $n/2$.

▶ **Theorem 7.** *For every $s > 0$ and constant $d > 0$, every $\mathrm{AC}^0[\oplus]$ circuit C of size s and depth d has a certifying polynomial of degree at most $n/2 - n/(\log s)^{\Theta(d)}$.*

We also show that this is essentially tight.

▶ **Lemma 8.** *For every $s > n^{\Omega(1)}$, there exist $\mathrm{AC}^0[\oplus]$ circuits C on n input bits with size s, such that every certifying polynomial for C has degree at least $n/2 - n/(\log s)^{\Theta(d)}$.*

These results are proved in Section 4. The proof of Theorem 7 uses the well-studied notion of approximating polynomials.

▶ **Definition 9** (ϵ-approximating polynomial)**.** An ϵ-approximating polynomial for a function f is a polynomial P such that $\Pr_{x \in \{0,1\}^n}[f(x) = P(x)] \geq 1 - \epsilon$.

The main ingredient in the proof of Theorem 7 is the following strengthening of Razborov's original theorem on approximating polynomials.

▶ **Lemma 10.** *For any $\epsilon \in (0, 1/2)$, any $\mathrm{AC}^0[\oplus]$ circuit C of size s and depth d has an ϵ-approximating polynomial of degree at most $(c \log s)^{d-1} \cdot (\log(1/\epsilon))$.*

We also show in Section 4 how Theorem 7 gives an alternate proof of Razborov's fundamental result that Majority does not have subexponential size $\mathrm{AC}^0[\oplus]$ circuits.

Finally, we state our lower bounds for the degree of approximating polynomials for $\mathrm{AC}^0[\oplus]$ circuits, showing the near-tightness of Lemma 10.

▶ **Theorem 11.** *For every $s, \epsilon > 0$, and every constant $d \geq 3$, there exist $\mathrm{AC}^0[\oplus]$ circuits C of size s and depth d such that for every polynomial P which is an ϵ-approximating polynomial for C, we have*

$$\deg(P) \geq \left(\log s - O(\log\log \frac{1}{\epsilon})\right)^{\Theta(d)} \cdot \log \frac{1}{\epsilon}.$$

The proof appears in Section 5.

3 Superlinear $\mathrm{AC}^0[\oplus]$ lower bounds for computing Approximate Majority

In this section, we prove Lemma 5 and Lemma 6, thus completing the proof of Theorem 4.

3.1 Linear-size $\mathrm{AC}^0[\oplus]$ circuits have low degree certifying polynomials

We now prove Lemma 5.

It will be more convenient to work with a certifying system of polynomials as opposed to a single certifying polynomial. Given a feasible system of polynomial equations over n variables $x_1, x_2, \ldots, x_n$, say

$$\begin{aligned} p_1(x) &= 0 \\ p_2(x) &= 0 \\ &\vdots \\ p_t(x) &= 0 \end{aligned}$$

we define the degree of the system to be $\sum_{i=1}^{t} \deg(p_i)$. Clearly, the set of solutions to the above system is exactly the set of roots of $1 - \prod_{i=1}^{t}(1 - p_i)$, which is a polynomial of degree at most $\sum_{i=1}^{t} \deg(p_i)$.

Given a feasible system of polynomial equations $\mathcal{P}$, we denote by $\mathrm{Sol}(\mathcal{P})$ the non-empty set of solutions of $\mathcal{P}$; when $\mathcal{P}$ sets just a single polynomial p, we denote use $\mathrm{Sol}(p)$ instead of $\mathrm{Sol}(\mathcal{P})$. By a *restriction*, we will mean simply a feasible system of polynomial equations.

Given a restriction $\mathcal{P}$ and a boolean circuit C, we will denote by $C|_{\mathcal{P}}$ the circuit C restricted to inputs from $\mathrm{Sol}(\mathcal{P})$. We say a gate g of the circuit C is *live* under the restriction given by $\mathcal{P}$ if g takes values 0 as well as 1 under inputs from $\mathrm{Sol}(\mathcal{P})$. Note that if a gate g is not live under a restriction, we can simplify the circuit C to a smaller circuit C' which computes the same function on the restricted inputs.

We say that a circuit C is *live* under the restriction $\mathcal{P}$ if every gate of C is live under $\mathcal{P}$. The above implies that, given any circuit C and restriction $\mathcal{P}$, there exists a live circuit C' of size at most the size of C that computes the same function as C on inputs from $\mathrm{Sol}(\mathcal{P})$.

Proof of Lemma 5. The proof will proceed as follows: after restricting the given circuit C to the roots of a well-chosen low-degree polynomial restriction $\mathcal{P}$, we will obtain an equivalent circuit C' that has the property that each of the AND and OR gates of C' have very small fan-in (say n^{ϵ} for $\epsilon \ll 1/d$). At this point, the entire circuit C' computes a low-degree polynomial p and by fixing p to a feasible value, we finish the proof of the lemma.

Say we have an increasing sequence of numbers $1 < D_1 < \ldots < D_d$ (we will fix the exact values of D_i ($i \geq 1$) later). We wish to obtain a restriction $\mathcal{P}$ under which C is equivalent to a circuit C' which has the property that every AND and OR gate at height i has fan-in at most D_i. It is easy to see that this implies that the function computed by C' is a polynomial of degree at most $D_1 D_2 \cdots D_d$.

We proceed to construct a suitable restriction $\mathcal{P}$ in d steps. After the ith step, we obtain a restriction $\mathcal{P}_i$ under which there is a circuit C_i of size at most s for which the above fan-in bound holds for all heights $j \leq i$. Assuming that the $(i-1)$th step has been completed, we describe how Step i is performed for $i \geq 1$. (Note that nothing needs to be done for height 0.)

We assume that C_i is live. Otherwise, we can obtain and work with an equivalent circuit that is of at most the size of C_i and satisfies the same fan-in restrictions as C_i. Let B_i denote the "bad" gates at height i: that is, the AND and OR gates at height i that have fan-in at least D_i. We use a basic subroutine $\mathrm{Fix}(i, C_i)$ that simplifies the circuit C_i by augmenting the restriction $\mathcal{P}_i$ as follows:

$\mathrm{Fix}(i, C_i)$: Since there are at least $|B_i|D_i$ wires between gates in B_i and lower levels (which contain at most s gates), there is some gate g at height less than i that is adjacent to $|B_i|D_i/s$ gates. By the fan-in restrictions on C_i, this gate computes a polynomial p_g of degree at most $D_1 \cdots D_{i-1}$ (the empty product in the case $i = 1$ is assumed to be 1). Moreover, since the circuit C_i is live, this gate can be set to both 0 and 1. We wish to add the restriction $p_g = 0$ or $p_g - 1 = 0$ to $\mathcal{P}_i$ corresponding to setting the gate to 0 or 1 respectively. Setting the gate g to 1 sets all the OR gates that g feeds into to 1 and setting g to 0 sets all the AND gates that g feeds into to 0. Hence, there is some setting that sets at least $|B_i|D_i/2s$ many gates in B_i to constant. We set the gate g to this boolean value.

Note that $\mathrm{Fix}(i, C_i)$ reduces the number of live bad gates to at most $|B_i|(1 - D_i/2s)$. We are now ready to describe Step i. Until the set of bad nodes B_i is empty, we repeatedly call the subroutine, $\mathrm{Fix}(i, C'_i)$ where C'_i represents the circuit we currently have. After an application of the subroutine $\mathrm{Fix}(i, C'_i)$ adds another equation to our current restriction $\mathcal{P}'_i$, we fix the non-live nodes and simplify the circuit until it becomes live again (this process, of course, does not increase the fan-in of any node). Note that since we are only fixing live nodes, the system of polynomial equations $\mathcal{P}'_i$ we maintain is feasible. Moreover, since the size of B_i is falling by a factor of at most $(1 - D_i/2s)$ after each application of $\mathrm{Fix}(i, C'_i)$ and $|B_i| \leq s \leq n^{O(1)}$, we need to apply $\mathrm{Fix}(i, C'_i)$ at most $\frac{2s \log |B_i|}{D_i} = O(s \log n/D_i)$ times to reduce B_i to the empty set.

Let us analyze the total degree of the equations added to the restriction during the ith step. Each equation added is a polynomial of degree at most $D_1 D_2 \cdots D_{i-1}$. Hence, the total degree of the added equations is $O(s \log n D_1 D_2 \cdots D_{i-1}/D_i)$.

At the end of Step d, we have a circuit C_d computing a polynomial of degree at most $D' = D_1 D_2 \cdots D_d$ that agrees with the original circuit C on a restriction of degree at most

$$D'' = O(s \log n) \left(\frac{1}{D_1} + \frac{D_1}{D_2} + \frac{D_1 D_2}{D_3} + \cdots + \frac{D_1 D_2 \cdots D_{d-1}}{D_d} \right)$$

We would like to set $D_1, \cdots, D_d$ such that both D' and D'' to be $o(n)$. We will choose K and the D_is such that $D_1 D_2 \cdots D_{i-1}/D_i = K$ for each i. This implies that $D_i = K^{2^{i-1}}$. Furthermore, we have $D' \leq K^{2^d}$ and $D'' \leq O(s \log n/K)$.

Setting $K = n^{1/(2^d+1)}$ and $\epsilon_d = \frac{1}{2^{d+1}}$, we get D' as well as D'' are $o(n)$ as long as $s \leq n^{1+\epsilon_d}$. Thus, by setting the polynomial p computed by the circuit C_d to some feasible value, we obtain a restriction of degree $D' + D'' = o(n)$ under which the circuit C becomes constant.

As mentioned above, this implies that there is a certifying polynomial for C of degree $o(n)$. ◀

3.2 Approximate Majority does not have any low degree certifying polynomials

We now prove Lemma 6.

Proof of Lemma 6. Let p be any polynomial of degree $d \leq a$ that takes the value 0 at some point of $\mathbb{F}_2^n$. We will show that it cannot be that f is constant on $\mathrm{Sol}(p)$.

Our intermediate claim is that $\mathrm{Sol}(p)$ intersects every Hamming ball of radius a. By translating p if necessary, we may assume that the Hamming ball is centered at the origin, and thus we seek to prove that there is a point of Hamming weight at most a where p vanishes.

Given the intermediate claim, it follows that there exist $x_0, x_1 \in \mathbb{F}_2^n$ with $p(x_0) = p(x_1) = 0$ such that the Hamming weight of x_0 is at most a, and the Hamming weight of x_1 is at least $n - a$. Thus f cannot be constant on $\mathrm{Sol}(p)$.

Now we prove the claim. Let $\tilde{p}$ denote the unique multilinear polynomial which agrees with p on $\mathbb{F}_2^n$. Since $\deg(p) \leq a$, we have $\deg(\tilde{p}) \leq a$. Now let q be the polynomial $1 - \tilde{p}$. Notice that q is multilinear and has degree at most a. Since $\mathrm{Sol}(p)$ is nonempty, we see that q is non-zero. Consider the monomials of q. Since $q \neq 0$, there must be a minimal $S \subseteq [n]$ (possibly empty) such that the monomial $\prod_{i \in S} X_i$ appears in q (i.e., has a non-zero coefficient) but no monomial $\prod_{i \in T} X_i$ for $T \subsetneq S$ appears in q. Let $x \in \mathbb{F}_2^n$ be the input that takes value 1 at exactly the indices in S. It is easy to see that $q(x) = 1$ and hence $\tilde{p}(x) = p(x) = 0$. Moreover, the Hamming weight of x is equal to the size of S which is at most $\deg(q) \leq a$. Hence, we see that $\mathrm{Sol}(p)$ does intersect the Hamming ball of radius a. This completes the proof of the claim, and hence the proof of Lemma 6.

◀

4 Certifying polynomials for general $\mathrm{AC}^0[\oplus]$ circuits

Given the results of the previous section, it makes sense to ask what are the lowest degree certifying polynomials we can obtain for general (i.e. significantly larger than linear-sized) $\mathrm{AC}^0[\oplus]$ circuits. Using an easy linear algebraic argument, it can be shown that *every* function, irrespective of its complexity, has a certifying polynomial of degree at most $n/2$ (and in this generality, it cannot be improved). In this section, we use Razborov's approximations for $\mathrm{AC}^0[\oplus]$ circuits by probabilistic polynomials to derive somewhat better certifying polynomials for functions with small $\mathrm{AC}^0[\oplus]$ circuits. In particular, we show that polynomial-sized $\mathrm{AC}^0[\oplus]$ circuits have certifying polynomials of degree $n/2 - n/(\log n)^{O(1)}$.

Though the improvement over the trivial $n/2$ bound above might seem small, the existence of such certifying polynomials is quite powerful: we demonstrate this by showing how this fact, along with Lemma 6, can be used to give a (slightly) conceptually different proof of Razborov's result that Majority does not have subexponential size $\mathrm{AC}^0[\oplus]$ circuits. We note that the proof is essentially unchanged at a technical level from the proofs of [9, 13], but the higher-order concepts involved seem curiously different. More specifically, this seems to provide a different 'constructive' (in the sense of Razborov and Rudich [10]) lower bound criterion for lower bounds against $\mathrm{AC}^0[\oplus]$ which is reminiscent of the work of Aspnes et al. [4].

The main theorem of this section is the following.

▶ Theorem 7 (Restated from Section 2). For every $s > 0$ and constant $d > 0$, every $\mathrm{AC}^0[\oplus]$ circuit C of size s and depth d has a certifying polynomial of degree at most $n/2 - n/(\log s)^{\Theta(d)}$.

The above theorem shows that functions computed by small subexponential size $\mathrm{AC}^0[\oplus]$ circuits have nontrivial certifying polynomials.

We will need to use probabilistic polynomials in the proof.

▶ **Definition 12** (Probabilistic polynomials)**.** An ϵ-error probabilistic polynomial of degree D for a function $f : \{0,1\}^n \to \{0,1\}$ is a random polynomial $\mathbf{P}$ of degree at most D (chosen according to some distribution over polynomials of degree at most D) such that for any $x \in \{0,1\}^n$, we have $\Pr_{\mathbf{P}}[f(x) = \mathbf{P}(x)] \geq 1 - \epsilon$.

Clearly, if a function f has an ϵ-error probabilistic polynomial $\mathbf{P}$ of degree D, then by averaging, it has an ϵ-approximating polynomial P of degree D as well.

We need the following well-known theorem, due to Razborov, on the existence of ϵ-error probabilistic polynomials for $\mathrm{AC}^0[\oplus]$. A node of a circuit C is said to be *internal* if it is not a leaf.

▶ **Theorem 13** (Razborov [9]). *For any $\epsilon \in (0, 1/2)$, any $\mathrm{AC}^0[\oplus]$ circuit C with at most s internal nodes and depth $d \geq 1$ has an ϵ-error probabilistic polynomial of degree at most $(\log(s/\epsilon))^d$. In particular, C has an ϵ-approximating polynomial of degree at most $(\log(s/\epsilon))^d$.*

Using Theorem 13 directly in our arguments would only give us a version of Theorem 7 with weaker parameters. To obtain the parameters mentioned above, we need a strengthening of Theorem 13 that does better for small ϵ. The proof follows quite simply from Razborov's theorem above, though to the best of our knowledge, this has not been observed in the literature.

▶ Lemma 10 (Restated from Section 2 in a stronger form). For any $\epsilon \in (0, 1/2)$, any $\mathrm{AC}^0[\oplus]$ circuit C of size s and depth d has an ϵ-error probabilistic polynomial of degree at most $(c \log s)^{d-1} \cdot (\log(1/\epsilon))$ for some absolute constant $c > 0$. In particular, C has an ϵ-approximating polynomial of degree at most $(c \log s)^{d-1} \cdot (\log(1/\epsilon))$.

Proof. Let C be an $\mathrm{AC}^0[\oplus]$ circuit of size s and depth d. Let g be the output gate of the circuit and let $C_1, \ldots, C_k$ ($k \leq s$) be the depth $d-1$ subcircuits of C feeding into g. By Theorem 13, we know that each C_i ($i \in [k]$) has a $(1/10s)$-approximating polynomial $\mathbf{P}_i$ of degree at most $(O(\log s))^{d-1}$. Also by Theorem 13, we know that the function computed by g has an $\mathrm{AC}^0[\oplus]$ circuit with just one *internal* node and hence has a $(1/10)$-approximating polynomial $\mathbf{P}$ of degree $O(1)$. The probabilistic polynomial $\mathbf{P'} := \mathbf{P}(\mathbf{P}_1, \ldots, \mathbf{P}_k)$ is a $1/5$-error probabilistic polynomial for C, since for any $x \in \{0,1\}^n$,

$$\begin{aligned}
\Pr_{\mathbf{P'}}[C(x) \neq \mathbf{P'}(x)] \leq & \Pr_{\mathbf{P}_1,\ldots,\mathbf{P}_k}[\exists i \in [k] :\ C_i(x) \neq \mathbf{P}_i(x)] + \\
& \Pr_{\mathbf{P}}[g(C_1(x), \ldots, C_k(x)) \neq \mathbf{P}(C_1(x), \ldots, C_k(x))] \\
\leq & \sum_{i \in [k]} \Pr_{\mathbf{P}_i}[C_i(x) \neq \mathbf{P}_i(x)] + \\
& \Pr_{\mathbf{P}}[g(C_1(x), \ldots, C_k(x)) \neq \mathbf{P}(C_1(x), \ldots, C_k(x))] \\
\leq & k/10s + 1/10 \leq 1/10 + 1/10 = 1/5
\end{aligned}$$

Note that $\mathbf{P'}$ has degree at most $(O(\log s))^{d-1}$. Let $\ell = c' \log(1/\epsilon)$ for a constant c' that we will choose later in the proof. Let $\mathbf{P}'_1, \ldots, \mathbf{P}'_\ell$ be ℓ independent copies of the probabilistic polynomial $\mathbf{P'}$. Let $\mathbf{Q}$ denote the probabilistic polynomial $\mathrm{Maj}(\mathbf{P}'_1, \ldots, \mathbf{P}'_\ell)$, where Maj is just the polynomial of degree at most ℓ that computes the majority of ℓ bits. Clearly, $\mathbf{Q}$ is of degree at most $(O(\log s))^{d-1} \cdot \ell = (O(\log s))^{d-1} \cdot \log(1/\epsilon)$. We claim that $\mathbf{Q}$ is an ϵ-error probabilistic polynomial for C, which will finish the proof of the corollary.

For any input $x \in \{0,1\}^n$, each $\mathbf{P}'_j(x)$ predicts the value of $C(x)$ correctly with probability $4/5$. Now, for $\mathbf{Q}(x)$ to predict $C(x)$ incorrectly, a *majority* of the $\mathbf{P}'_j$ ($j \in [\ell]$) must predict the value of $C(x)$ incorrectly and by a Chernoff bound, the probability of this is bounded by $\exp\{-\Omega(\ell)\}$, which is at most ϵ for a large enough constant $c' > 0$. ◀

The next lemma shows that functions with low-degree ϵ-approximating polynomials also have low-degree certifying polynomials.

▶ **Lemma 14.** *Suppose $f : \{0,1\}^n \to \{0,1\}$ has a degree D ϵ-approximating polynomial. Then f has a certifying polynomial of degree at most $\frac{n}{2} - c_1\sqrt{n \log \frac{1}{\epsilon}} + D$, where c_1 is an absolute constant.*

Proof. Let P be the given ϵ-approximating polynomial. Let S be the set of points where P differs from f. We have $|S| \leq \epsilon \cdot 2^n$.

Let D_0 be the smallest integer such that

$$\binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{D_0} > |S|.$$

By linear algebra, there is a non-zero polynomial Q of degree at most D_0 that vanishes on S. Note that one of $Q \cdot P$ and $Q \cdot (1-P)$ is a non-zero polynomial. Moreover, for any input x s.t. $Q(x) = 1$, $P(x) = f(x)$.

Thus, it follows that one of $1 - Q \cdot P$ or $1 - Q \cdot (1-P)$ is a certifying polynomial for f with degree at most $D_0 + D$ (provided $D_0 + D < n$; if not then the result is vacuously true). To finish the proof, we note that $D_0 \leq \frac{n}{2} - c_1\sqrt{n \log \frac{1}{\epsilon}}$. ◀

Proof of Theorem 7. Combining Lemma 10 and Lemma 14, we conclude that for every $\epsilon > 0$. C has a certifying polynomial of degree at most

$$\frac{n}{2} - c_1\sqrt{n \cdot \log \frac{1}{\epsilon}} + (c_2 \log s)^{d-1} \cdot \log(1/\epsilon),$$

where $c_1, c_2 > 0$ are absolute constants. In particular, setting $\epsilon = \exp\{-n/(\log s)^{\Theta(d)}\}$ above, we get that C has a certifying polynomial of degree at most $n/2 - n/(\log s)^{\Theta(d)}$. ◀

Combining Theorem 7 with Lemma 6 (and using the fact that Majority is an $(n/2 - 1, n/2+1)$ Approximate Majority), we get an alternate proof of the fact that Majority cannot be computed by $\mathrm{AC}^0[\oplus]$ circuits of size smaller than $\exp(n^{\Omega(1/d)})$.

Finally, we show that the bound of Theorem 7 is essentially tight.

▶ **Lemma 15** (Restated from Section 2). *For every $s > n^{\Omega(1)}$, there exist $\mathrm{AC}^0[\oplus]$ circuits C on n input bits with size s, such that every certifying polynomial for C has degree at least $n/2 - n/(\log s)^{\Theta(d)}$.*

Proof. Let δ be a parameter to be specified later. Let C be the $\mathrm{AC}^0[\oplus]$ circuit for $((1/2 - \delta)n, (1/2+\delta)n)$ Approximate Majority given by Theorem 3. Then we have $|C| = 2^{(1/\delta)^{O(\frac{1}{d})}} \cdot n^{O(1)}$. We choose δ so that $|C| = s$; this gives $\delta = \frac{1}{(\log s - c \log n)^{\Omega(d)}}$.

By Lemma 6, any certifying polynomial for C has degree at least $n \cdot (\frac{1}{2} - \delta) = \frac{n}{2} - \frac{n}{(\log s)^{\Theta(d)}}$. ◀

5 Lower bounds for approximating polynomials

We now use the tools of the previous two sections to show near-optimal lower bounds on the degree of approximating polynomials for $\mathrm{AC}^0[\oplus]$ circuits. It is a folklore fact that ϵ-approximations for $\mathrm{AC}^0[\oplus]$ circuits of size s and depth d are required to have degree at least $\max\{(\log s)^{\Omega(d)}, \log(1/\epsilon)\}$. In this section, we show a stronger lower bound of $\Theta((\log s)^{\Omega(d)} \cdot \log(1/\epsilon))$, which essentially matches the upper bound obtained in Lemma 10. Our lower bound example is just a suitable Approximate Majority and thus holds even for AC^0 circuits.

We prove the lower bound by exploiting Lemma 14 in the contrapositive. Since there are Approximate Majorities that are efficiently computable in AC^0, by Lemma 6, we know that AC^0 circuits can compute functions that do not have efficient certifying polynomials. We can then use Lemma 14 to infer a lower bound on the degree of ϵ-approximations to AC^0 circuits.

▶ Theorem 11 (Restated from Section 2). For every $s, \epsilon > 0$, and every constant $d \geq 3$, there exist $\mathrm{AC}^0[\oplus]$ circuits C of size s and depth d such that for every polynomial P which is an ϵ-approximating polynomial for C, we have

$$\deg(P) \geq \left(\log s - O(\log\log \frac{1}{\epsilon})\right)^{\Theta(d)} \cdot \log \frac{1}{\epsilon}.$$

Proof. Let δ and m be parameters (to be specified later). Let C be an $\mathrm{AC}^0[\oplus]$ circuit on m inputs which computes a $((\frac{1}{2}-\delta)m, (\frac{1}{2}+\delta)m)$-approximate majority. By Theorem 3, such an AC^0 circuit can be taken to have depth d and size at most $2^{(1/\delta)^{O(\frac{1}{d})}} \cdot m^{O(1)}$. We will choose m and δ so that this size equals s.

Suppose P is an ϵ-approximating polynomial for C with degree D. By Lemma 14, there is a degree $\frac{m}{2} - c_1\sqrt{m \log \frac{1}{\epsilon}} + D$ polynomial Q which is a certifying polynomial for C.

But since C is a $((\frac{1}{2}-\delta)m, (\frac{1}{2}+\delta)m)$ Approximate Majority, Lemma 6 tells us that that $\deg(Q) \geq (\frac{1}{2}-\delta)\cdot m$.

Putting this together, we get that $D \geq c_1\sqrt{m \log \frac{1}{\epsilon}} - \delta \cdot m$.

We now choose m, δ so that $c_1\sqrt{m \log \frac{1}{\epsilon}} = 2\delta \cdot m$ and $s = 2^{(1/\delta)^{O(\frac{1}{d})}} \cdot m^{O(1)}$. Thus:

$$m = \left(\log s - O(\log\log \frac{1}{\epsilon})\right)^{\Theta(d)} \cdot \log \frac{1}{\epsilon}.$$

We therefore get

$$D \geq \left(\log s - O(\log\log \frac{1}{\epsilon})\right)^{\Theta(d)} \cdot \log \frac{1}{\epsilon},$$

as desired. ◀

6 Discussion and Open Questions

We have seen that certifying polynomials are a natural and useful notion in the context of lower bounds for $\mathrm{AC}^0[\oplus]$ circuits. We also saw that they have a rather interesting interaction with the well-studied notion of approximating polynomials for $\mathrm{AC}^0[\oplus]$ circuits.

The fundamental question we would like to answer is whether we can prove a size-hierarchy theorem for $\mathrm{AC}^0[\oplus]$ analogous to the results of Rossman [11] and Amano [3] for AC^0. It would even be interesting to obtain the weaker separation of uniform $\mathrm{AC}^0[\oplus]$ circuits of size $n^{\log n}$ from polynomial-sized $\mathrm{AC}^0[\oplus]$ circuits? Good candidates for proving these separations seem to be the parity of the number of k-cliques in a graph for the former, and the elementary symmetric polynomial of degree $\log n$ for the latter. We have taken the first step in this direction by demonstrating a function that has polynomial-sized uniform AC^0 circuits but not near-linear sized $\mathrm{AC}^0[\oplus]$ circuits.

Another question that we leave open is to prove lower bounds on the degree for ϵ-approximating polynomials for depth 2 $\mathrm{AC}^0[\oplus]$ circuits. Our lower bound utilized small $\mathrm{AC}^0[\oplus]$ circuits for Approximate Majority, which only exist for depth 3 and higher.

It would be interesting to see whether certifying objects (analogous to the certifying polynomials studied here) exist for other, more powerful, circuit classes, and if they can be used to prove new circuit lower bounds.

Acknowledgements

We would like to thank Albert Atserias for asking us about the tradeoff between degree and error for approximating polynomials for $AC^0[\oplus]$ circuits. We would also like to thank the reviewers of FSTTCS 2012 for pointing out some errors and improving the quality of the exposition.

References

1 Miklós Ajtai. *Approximate counting with uniform constant-depth circuits.*, pages 1–20. Providence, RI: American Mathematical Society, 1993.
2 Miklós Ajtai and Michael Ben-Or. A theorem on probabilistic constant depth computations. In *STOC*, pages 471–474, 1984.
3 Kazuyuki Amano. k-subgraph isomorphism on AC^0 circuits. *Computational Complexity*, 19(2):183–210, 2010.
4 James Aspnes, Richard Beigel, Merrick L. Furst, and Steven Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):135–148, 1994.
5 Shiva Chaudhuri and Jaikumar Radhakrishnan. Deterministic restrictions in circuit complexity. In *STOC*, pages 30–36, 1996.
6 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.
7 Johan Håstad. *Computational limitations of small-depth circuits.* The MIT Press, Cambridge(MA)-London, 1987.
8 Prabhakar Ragde and Avi Wigderson. Linear-size constant-depth polylog-treshold circuits. *Inf. Process. Lett.*, 39(3):143–146, 1991.
9 Alexander A. Razborov. Lower bounds on the size of constant-depth networks over a complete basis with logical addition. *Mathematicheskie Zametki*, 41(4):598–607, 1987.
10 Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.
11 Benjamin Rossman. On the constant-depth complexity of k-clique. In *STOC*, pages 721–730, 2008.
12 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 77–82, 1987.
13 Roman Smolensky. On representations by low-degree polynomials. In *FOCS*, pages 130–138, 1993.
14 Emanuele Viola. On approximate majority and probabilistic time. *Computational Complexity*, 18(3):337–375, 2009.
15 Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.

Randomly-oriented k-d Trees Adapt to Intrinsic Dimension

Santosh S. Vempala*

School of Computer Science
Georgia Tech
vempala@gatech.edu

Abstract

The classic k-d tree data structure continues to be widely used in spite of its vulnerability to the so-called curse of dimensionality. Here we provide a rigorous explanation: for randomly rotated data, a k-d tree adapts to the intrinsic dimension of the data and is not affected by the ambient dimension, thus keeping the data structure efficient for objects such as low-dimensional manifolds and sparse data. The main insight of the analysis can be used as an algorithmic pre-processing step to realize the same benefit: rotate the data randomly; then build a k-d tree. Our work can be seen as a refinement of *Random Projection trees* [7], which also adapt to intrinsic dimension but incur higher traversal costs as the resulting cells are polyhedra and not cuboids. Using k-d trees after a random rotation results in cells that are cuboids, thus preserving the traversal efficiency of standard k-d trees.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, E.1 Data Structures, G.3 Probability and Statistics

Keywords and phrases Data structures, Nearest Neighbors, Intrinsic Dimension, k-d Tree

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.48

1 Introduction

The k-d tree, introduced by Bentley [4], is a classic data structure for nearest neighbor search. Roughly speaking, a k-d tree is constructed by recursively partitioning space using axis-parallel cuts, with each cut placed at the median of the point set along some axis. It is widely used in machine learning, computer vision, bioinformatics, astronomy and other fields.

For the ubiquitous nearest neighbor problem, k-d trees are the method of choice. Although other more sophisticated algorithms have been proposed in the nearly forty years since the invention of k-d trees, they remain the default approach to nearest neighbor problems in a variety of settings. They are space efficient, being only linear in size with respect to the number of points, and they are easy to construct and query.

Although they are so popular, k-d trees do have a major weakness. As the dimension n becomes large, in the worst case, nearest neighbor queries can take time close to linear in the total number of points (a manifestation of the "curse of dimensionality"). Thus our current state of knowledge is that k-d trees are an efficient heuristic approach in low dimension (without precise knowledge of running times) and their performance can degrade significantly as the dimension increases.

To overcome the the challenge of high dimensionality, researchers have designed other data structures for nearest neighbor search. These include tree-based structures such as

* Supported in part by NSF award AF-0915903

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 48–57

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

approximate Voronoi diagrams [12], cover trees [5], PCA trees [19] and navigating nets [17]. Another algorithmic solution is based on random projection [18], inspired by the Johnson-Lindenstrauss Lemma [15]. This idea was subsequently developed by applying a series of random projections together as *locality-sensitive* hash functions [14, 13, 1], leading to the strongest known upper bounds for *approximate* nearest neighbor search. Roughly speaking, these bounds allow for fixed polynomial (slightly superlinear) storage space and sublinear query time, in return for constant-factor approximations to the nearest neighbor distance.

A different line of research has focused on exploiting some form of low-dimensional structure in the data, e.g., cover trees [5] and random projection trees [7, 10]. The idea here is that the ambient dimension is not the critical factor in the complexity, but rather some much smaller quantity that corresponds to the intrinsic dimension of the data set. An important notion of intrinsic dimension is the *Assouad* (or *doubling*) dimension [11], a slight variant of a concept defined by Assouad [3]. Let $B(x, r)$ be the closed ball of radius r centered at x.

▶ **Definition 1.** The *doubling dimension* of a set $S \subseteq \mathbb{R}^n$ is the smallest integer d s.t. for every $x \in \mathbb{R}^n$ and $r > 0$, $B(x, r) \cap S$ can be covered by at most 2^d balls of radius $r/2$.

An interval in $\mathbb{R}$ has doubling dimension 1. Any subset of $\mathbb{R}^k$ has doubling dimension $O(k)$; so subsets restricted to k-dimensional subspaces in $\mathbb{R}^n$ have doubling dimension $O(k)$. The union over $1 \leq i \leq n$ of the intervals $[-e_i, e_i]$ has doubling dimension $\log n$ (where $e_1, \ldots, e_n$ are the canonical unit vectors). This sparse example can be generalized — if every point in $S \subset \mathbb{R}^n$ has at most d nonzero coordinates, then the doubling dimension of S is $O(d \log n)$ [6].

Dasgupta and Freund's random projection trees (RP trees) are built as follows: pick a random direction at every partitioning step, independently for each cell, and split the current cell at a random point within a small interval of the median of the current data points. These trees have the property that the diameter of cells in the data structure decreases quickly with the number of splits – it takes roughly $O(d \log d)$ splits to halve the diameter where d is the intrinsic dimension. Dasgupta and Freund termed this behavior "adapting to Assouad dimension". Subsequently, the RP tree has found applications in other settings including tree-based vector quantization [8] and regression [16]. The random directions used to build an RP tree are not orthogonal to each other and at each level of the tree, there are many different cuts used, leading to a data structure whose cells are general polyhedra rather than cuboids as in standard k-d trees. One advantage of the standard k-d tree that is lost here is that traversing the tree only needs comparison on a single coordinate at a time, while for RP trees this goes up by at least a factor of n (since one has to compare along a random direction). Moreover, computing the distance of a point to a cell, which is now a general polyhedron, is substantially more complicated. Indeed, their original paper does not give a nearest neighbor algorithm.

On the other hand, k-d trees do not have a nice dependence on doubling dimension while RP trees do. This is seen in the example of a points distributed along n orthogonal lines, one parallel to each axis. In this example, halving the diameter requires n splits, i.e., depth n, even though the doubling dimension is only $\log(2n)$.

In this paper, we propose a conceptually simple and algorithmically efficient variant of k-d trees that adapts to intrinsic dimension. In fact, our algorithm is essentially a pre-processing step for a k-d tree. The preprocessing consists of a *random rotation* of the ambient space, i.e., instead of the standard basis for constructing a tree, we use a random basis. The overhead in the running time could be negligible as the database would be rotated in advance, and a query point only has to be mapped once to the chosen basis before the search is carried out.

Our main theorem asserts that such a transformation leads to a strong guarantee for k-d trees, namely that they adapt to the intrinsic dimension, in the same way that RP trees do. The Randomized k-d tree algorithm is described precisely in the next section.

▶ **Theorem 2** (Main). *Let $S \subset \mathbb{R}^n$ be a finite set with m points and doubling dimension d. Assuming $d \log d \leq c_0 n$, for the Randomized k-d tree, with probability at least $1 - me^{-c_1 n}$, for any cell C of the tree and every cell C' that is at least $c_2 d \log d$ levels below C, we have*

$$\text{diam}(C' \cap S) \leq \frac{1}{2}\text{diam}(C \cap S)$$

where c_0, c_1, c_2 are absolute constants.

In other words, when the doubling dimension is low, it takes only a small number of rounds of splits to halve the diameter of cells. A stronger guarantee would be to give an actual bound on the query time based on doubling dimension. However, we are not aware of such a connection between cell diameter and query times.

This theorem also provides an intriguing perspective on the standard use of k-d trees showing that simply taking a random rotation of the data could yield a configuration of the dataset more amenable to nearest neighbor search via k-d trees. Alternatively, one can also view this result as an explanation of the success of k-d trees, namely if one assumes that the basis in which measurements are made is essentially random, then these trees adapt to the intrinsic dimension, i.e., they work well on average, if one views the data as coming from a randomly chosen basis. Both these perspectives are supportive of the idea that k-d trees are actually an excellent choice whenever the intrinsic dimension is significantly lower than the ambient dimension.

We note that this is a technically simple paper, based heavily on techniques from the literature. The main obstacle we overcome in the analysis is that the splits used in our method are not independent, unlike RP trees where each splitting direction is chosen independently of all others. As far as we know, the simple idea of a random rotation in advance does provide the first reasonable explanation of the performance of k-d trees with increasing dimension on real data sets (as they might have low Assouad dimension). Moreover, the insight of the analysis can be made algorithmic: rotate the data randomly; then build a k-d tree. It remains to be seen whether this pre-processing step is useful in practice.

2 Algorithm

As mentioned in the introduction, our algorithm is the following: we pick a random orthogonal basis for space and then build a k-d tree using this basis. The only change we make is that we make is that instead of splitting at the median when we partition a cell, we split at a random point in an interval around the median (this modification was also used by Dasgupta and Freund [7]). It is conceivable that perturbation near the median can be avoided by adding some random points to the data before building the tree; we do not explore this here.

3 Analysis

3.1 Outline

Our goal is to show that any subset S of bounded diameter will be partitioned into cells of at most half the diameter within $O(d \log d)$ levels of partitioning applied to the subset. To prove this, we first cover S with balls of significantly smaller diameter, then show that

Randomized k-d Tree.
1. Pick a random basis $V = \{v_1, \ldots v_n\}$ of $\mathbb{R}^n$.
2. Run KD-Tree$(S, V, 1)$.

KD-Tree(S, V, i)**.**
- If $|S| = 1$, return S.
  1. Let 2Δ be the diameter of S.
  2. Let m be the median of S along v_i and δ be uniform random in $\left[-\frac{6\Delta}{\sqrt{n}}, \frac{6\Delta}{\sqrt{n}}\right]$.
  3. $S^- = \{x \in S : \langle x, v_i \rangle \leq m + \delta\}$; $S^+ = S \setminus S^-$.
  4. $T^- =$ KD-Tree$(S^-, V, i \mod n + 1)$; $T^+ =$ KD-Tree$(S^+, V, i \mod n + 1)$.
  5. Return $[T^-, T^+]$.

Figure 1 Randomized k-d Tree Algorithm

with good probability, our partitioning procedure separates any pair of balls that are far enough part into different cells within $O(d \log d)$ levels of partitioning. The cells obtained at the end of this process will have the claimed diameter bound. Dasgupta and Freund use random independent splits for each cell, and a union bound for the failure probability. In our case, the splits come from a single basis and the same split direction is applied to all cells at one level, so we have to analyze the resulting conditioning and dependencies. RP trees pick a completely random direction to make the next split, our trees pick the next vector in the random basis. To argue that the latter achieves similar performance, we observe that a random basis can be chosen by picking a random unit vector, then a random unit vector orthogonal to it, and so on, each time picking a random unit vector orthogonal to the span of the vectors chosen so far. Our analysis idea is to consider the projection orthogonal to all basis vectors used for cuts so far and argue that this projection does not collapse points or shrink balls too much as long as not too many vectors have been chosen. Once we condition on this, a random vector in the remaining subspace is almost as good as a random vector in the full space.

It is, however, necessary that we incur some dependence on the number of points, since we are picking only a fixed basis, i.e., the total randomness is bounded. We could set up a large enough point set such that for any chosen basis, eventually we reach a cell that takes much more than $O(d \log d)$ cuts to halve in diameter. We get around this issue by assuming that the total number of points is at most exponential in the ambient dimension, i.e., at most 2^{cn} for some constant c.

3.2 Preliminaries

Our main tool is the Johnson-Lindenstrauss Lemma [15]. For a subspace V of $\mathbb{R}^n$, let $\pi_V(.)$ denote orthogonal projection to the subspace V. We will use the following version from [2, 20] (see also [9, 13]).

▶ **Lemma 3.** *Fix a unit vector $u \in \mathbb{R}^n$, let V be a random k dimensional subspace where $k < n$, and $\epsilon > 0$ then:*

$$\Pr\left(\|\pi_V(u)\|^2 > (1+\epsilon)\frac{k}{n}\right) \leq e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}$$

$$\Pr\left(\|\pi_V(u)\|^2 < (1-\epsilon)\frac{k}{n}\right) \leq e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}.$$

As a direct implication, for any finite set of points S in $\mathbb{R}^n$, with probability at least

$$1 - 2\binom{|S|}{2} e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)},$$

we have

$$\forall u, v \in S, (1-\epsilon)\frac{k}{n}\|u-v\|^2 < \|\pi_V(u-v)\|^2 < (1+\epsilon)\frac{k}{n}\|u-v\|^2.$$

We also use the following standard bounds for $k = 1$.

▶ **Lemma 4.** *Let $u \in \mathbb{R}^n$ and $v \in \mathbb{R}^n$ be a random unit vector. For any $\beta > 0$,*

$$\begin{aligned}
\mathsf{Pr}\left(\|\pi_v(u)\| > \frac{\beta}{\sqrt{n}}\|u\|\right) &\leq \frac{2}{\beta}e^{-\frac{\beta^2}{2}} \\
\mathsf{Pr}\left(\|\pi_v(u)\| \leq \frac{\beta}{\sqrt{n}}\|u\|\right) &\leq \alpha\sqrt{\frac{2}{\pi}}.
\end{aligned}$$

3.3 Projection properties

The next lemma is a structural property that uses the doubling dimension, and is similar to what was shown in [7] for RP trees.

▶ **Lemma 5.** *Let $S \subset B(x, r)$ be a set of doubling dimension d. Let V be an arbitrary k-dimensional subspace of $\mathbb{R}^n$, v be a random unit vector orthogonal to V, $1 > \delta > 0$ and*

$$r' = \frac{3r}{\sqrt{n-k}}\sqrt{2\left(d + \log(2/\delta)\right)}$$

Then $\pi_v(S) \subseteq [\pi_v(x) - r', \pi_v(x) + r']$ with probability at least $1 - \delta$.

Proof. We consider a projection orthogonal to the given subspace V first, then a projection to a random vector in this subspace $W = V^\perp$. Since W has sufficiently large dimension, this will be nearly as good as projecting to a random vector in the full space.

Let C_1 be a minimum cover of S consisting of balls of radius $r/2$. From the definition of doubling dimension, C_1 has at most 2^d elements. Similarly C_2 will be a cover of $C_1 \cap S$ with balls of radius $r/4$; at level i, C_i will be cover of $C_{i-1} \cap S$ using at most 2^d balls of radius $r/2^i$ for each element of C_{i-1}.

Fix a ball $B(c, r/2^i)$ at level i and consider one of the balls, $B(c', r/2^{i+1})$, which covers it. Let $W = V^\perp$. We have for c and c', the center of these balls that

$$\|c - c'\| \leq \frac{r}{2^i}.$$

Next we compute the following:

$$\begin{aligned}
\mathsf{Pr}\left(\|\pi_v(c-c')\| \geq \beta\frac{r}{2^i}\sqrt{\frac{i+1}{n-k}}\right) &\leq \mathsf{Pr}\left(\|\pi_v(c-c')\| \geq \beta\frac{\|\pi_W(c-c')\|}{\sqrt{n-k}}\sqrt{i+1}\right) \\
&= \mathsf{Pr}\left(\|\pi_v(\pi_W(c-c'))\| \geq \beta\frac{\|\pi_W(c-c')\|}{\sqrt{n-k}}\sqrt{i+1}\right) \\
&\leq \frac{2}{\beta\sqrt{i+1}}e^{-\frac{\beta^2}{2}(i+1)} \\
&\leq \frac{\delta}{\beta}\left(\frac{\delta}{2}\right)^i e^{-d(i+1)}
\end{aligned}$$

where $\beta = \sqrt{2(d + \log(2/\delta))}$. Now we take a union bound over all balls used in covers at all levels. For level i, there are $2^{id}2^d = 2^{(i+1)d}$ pairs we need to consider. Thus, via a standard chaining argument a la Dudley,

$$\begin{aligned}\Pr\left(\exists c, c' \text{st } \|\pi_v(c-c')\| \geq \beta\frac{r}{2^i}\sqrt{\frac{i+1}{n-k}}\right) &\leq \sum_{i=0}^{\infty} 2^{(i+1)d}\frac{\delta}{\beta}\left(\frac{\delta}{2}\right)^i e^{-d(i+1)} \\ &\leq \frac{\delta}{\beta}\frac{1}{1-\delta/2} \\ &\leq \delta.\end{aligned}$$

So with probability at least $1-\delta$, every point $y \in S$ satisfies

$$\|\pi_v(y) - \pi_v(x)\| \leq \frac{\beta r}{\sqrt{n-k}}\sum_{i=0}^{\infty}\frac{\sqrt{i+1}}{2^i} \leq \frac{3r}{\sqrt{n-k}}\sqrt{2\left(d+\log(2/\delta)\right)}.$$

◀

The next lemma is from [7].

▶ **Lemma 6.** *For $S \subset B(x,\Delta)$, $\delta \in (0, 2/e^2]$ and a random unit vector $v \in \mathbb{R}^n$, with probability at least $1-\delta$,*

$$\|\text{median}(\pi_v(S)) - \pi_v(x)\| \leq \Delta\sqrt{\frac{2\log(2/\delta)}{n}}$$

▶ **Lemma 7.** *Let $S \subseteq B(x,\Delta)$ and $z \in B(x,\Delta)$. Let V be a k-dimensional subspace of $\mathbb{R}^n$ with $k < n/9$ and v be a random unit vector orthogonal to V. Then, with probability at least* 0.95,

$$\|\text{median}(\pi_v(S)) - \pi_v(z)\| \leq \frac{6\Delta}{\sqrt{n}}$$

Proof. By the triangle inequality, it suffices to show that $\|\pi_v(z-x)\| \leq 3\Delta/\sqrt{n}$ and that $\|\text{median}(\pi_v(S)) - \pi_v(x)\| \leq 3\Delta/\sqrt{n}$. The first bound uses Lemma 4 setting $\beta = \sqrt{8}$:

$$\Pr\left(\|\pi_v(z-x)\| \geq \beta\frac{\|z-x\|}{\sqrt{n-k}}\right) \leq \frac{2}{\beta}e^{-\frac{\beta^2}{2}} \leq \frac{1}{\sqrt{2}e^4}.$$

Next, since $k < n/9$,

$$\beta\frac{\|z-x\|}{\sqrt{n-k}} \leq 3\frac{\|z-x\|}{\sqrt{n}}.$$

Therefore,

$$\Pr\left(\|\pi_v(z-x)\| \geq 3\frac{\|z-x\|}{\sqrt{n}}\right) \leq \frac{1}{\sqrt{2}e^4}.$$

The second inequality is derived using Lemma 6 with $\delta = 2/e^4$.

$$\begin{aligned}\|\text{median}(\pi_v(S)) - \pi_v(x)\| &\leq \frac{\Delta}{\sqrt{n-k}}\sqrt{2\log\left(\frac{2}{\delta}\right)} \\ &\leq \frac{3\Delta}{\sqrt{n}}.\end{aligned}$$

Putting these inequalities together completes the proof, with a total failure probability of at most $(1/\sqrt{2}e^4) + (2/e^4) < 1/20$. ◀

3.4 Proof of Main Theorem

We are now ready to prove the main theorem. Let S be a set of points contained in a cell C of the tree with $\Delta = \operatorname{diam}(C \cap S)$, i,e., $S \subseteq B(x, \Delta)$.

Since S has doubling dimension d, we can cover it using $100d^{d/2}$ balls of radius $r = \Delta/100\sqrt{d}$.

Let $k < c_0 n$ and $\{v_1, \ldots, v_n\}$ be a set of random orthonormal vectors with $W = \operatorname{span}(v_1, \ldots, v_k)^{\perp}$. By by Lemma 3 and the remark following it, we have that for all centers u and v of our ball cover (including the center x) at all of at most m nodes of the tree,

$$\begin{aligned}
&\mathsf{Pr}\left(\forall u, v : \frac{9}{10}\frac{\|u-v\|^2(n-k)}{n} < \|\pi_W(u-v)\|^2 < \frac{11}{10}\frac{\|u-v\|^2(n-k)}{n}\right) \\
&\geq 1 - 10^4 d^d m e^{-\frac{n-k}{4}\left(\frac{1}{100}-\frac{1}{1000}\right)} \\
&\geq 1 - 10^4 m e^{c_0 n} e^{-\frac{n}{500}} \\
&\geq 1 - 10^4 m e^{-\frac{n}{1000}}
\end{aligned}$$

with $c_0 \leq 1/1000$. We will assume that this distortion bound holds for the rest of the proof.

Now consider two balls in this cover, $B = B(z, r)$ and $B' = B(z', r)$ where $z, z' \in B_(x, \Delta)$ and are more than $\Delta/2 - r$ apart. For each split there are three possibilities: either the partition separates B and B' (which we call a "good split"), or it intersects both B and B' (a "bad split") or the partition only intersects one or none of the two balls ("neutral split"). In the case of a "bad split", we have to now separate the four parts of B and B', and in the case of a "neutral split", we still only have to separate two objects. We will bound these probabilities for single steps in Lemmas 8 and 9 (whose proofs we defer to the end of this section).

▶ **Lemma 8** (Good splits). *Let $S \subset B(x, \Delta) \subset \mathbb{R}^n$ have doubling dimension d. Fix two balls $B(z, r)$ and $B(z', r)$ and a subspace V of dimension $k \leq n/9$ where:*

1. $z, z' \in B(x, \Delta)$.
2. $\|z - z'\| \geq \Delta/2 - r$.
3. $r \leq \Delta/(100\sqrt{d})$.
4. *The squared distances between x, z and z' are distorted by at most $1/10$ in $V^{\perp}$.*

Let v be a random unit vector orthogonal to V, and s be a point uniformly at random in the interval

$$[\operatorname{median}(\pi_v(S)) - 6\Delta/\sqrt{n}, \operatorname{median}(\pi_v(S)) + 6\Delta/\sqrt{n}].$$

Then with probability at least $1/200$, $\pi_v(B)$ and $\pi_v(B')$ lie on different sides of s.

▶ **Lemma 9** (Bad splits). *Under the above hypotheses of Lemma 8, then probability at most $1/300$, s is contained in the supports of $\pi_v(B)$ and $\pi_v(B')$.*

We prove these lemmas at the end of this section. To complete the mian proof, following [7], let p_i, be the probability that B and B' share a cell after i levels, i.e., they are not completely separated. Clearly, $p_1 \leq 199/200$ using Lemma 8. Moreover,

$$\begin{aligned}
p_i &\leq \mathsf{Pr}(\text{good split}) \times 0 + 2p_{i-1}\mathsf{Pr}(\text{bad split}) + \mathsf{Pr}(\text{neutral split})\, p_{i-1} \\
&\leq \frac{1}{200}\cdot 0 + \frac{2}{300}p_{i-1} + \left(1 - \frac{1}{200} - \frac{1}{300}\right)p_{i-1} \\
&\leq \frac{599}{600}p_{i-1}.
\end{aligned}$$

Thus we have p_k as being exponentially small in k. Denote $\alpha = 599/600$. If we take:

$$k = \frac{1}{\log(\alpha)}\left(d\log d + 5\log 10\right)$$

rounds of partitioning, then each pair of balls is in the same cell with probability at most $1/(10^5 d^d)$. Hence, by taking a union bound over all pairs of balls, no pair is in the same partition with probability at least $9/10$.

To extend the analysis to the entire tree, we simply note that with m points in S, there are at most m covers (one for each internal node of the tree) of $10^4 d^{d/2}$ balls where we have to preserve the distances between the centers.

We conclude this section with the proofs of the claims regarding good and bad splits.

Proof of Lemma 8. In Lemma 5, if we take $\delta = 2/e^9$ and $r \leq \Delta/100\sqrt{d}$, then we find that $\pi_v(B)$ lies in an interval of radius $\frac{3r}{\sqrt{n-k}}\sqrt{2(d+\log(2/\delta))}$:

$$\begin{aligned}\frac{3r}{\sqrt{n-k}}\sqrt{2(d+\log(2/\delta))} &\leq \frac{3\Delta}{100}\sqrt{\frac{2(d+9)}{d(n-k)}} \\ &\leq \frac{\Delta}{16\sqrt{n-k}}\end{aligned}$$

Next we show that with good probability,

$$\|\pi_v(z-z')\| \geq \Delta/(4\sqrt{n-k}).$$

To see this, note that projecting $z-z'$ to v is equivalent to projecting $\pi_W(z-z')$ to a random unit vector in W. We can apply Lemma 4 with $\beta = \sqrt{\frac{10}{9}}/4$

$$\begin{aligned}\mathsf{Pr}\left(\|\pi_v\pi_W(z-z')\| \leq \beta\frac{\|\pi_W(z-z')\|}{\sqrt{n-k}}\right) &\leq \beta\sqrt{\frac{2}{\pi}} \\ \mathsf{Pr}\left(\|\pi_v\pi_W(z-z')\| \leq \beta\sqrt{9/10}\frac{\|z-z'\|}{\sqrt{n-k}}\right) &\leq \beta\sqrt{\frac{2}{\pi}} \\ \mathsf{Pr}\left(\|\pi_v\pi_W(z-z')\| \leq \frac{\Delta}{4\sqrt{n-k}}\right) &\leq \sqrt{\frac{20}{\pi}}\frac{1}{12} < 0.42.\end{aligned}$$

Thus, with probability at least 0.68, there is a gap of at least

$$\frac{\Delta}{4\sqrt{n-k}} - 2\frac{\Delta}{16\sqrt{n-k}} \geq \frac{\Delta}{8\sqrt{n}}$$

between $\pi_v(B)$ and $\pi_v(B')$. On the other hand, by Lemma 7 applied to the centers of these balls, with probability at least 0.9, they are both within $6\Delta/\sqrt{n}$ of the median of the projection. Thus, with probability greater than $1/2$, we have both events: a large gap between the balls and both balls intersecting the interval around the median. The probability that random partition hits this gap, conditioned on these events is:

$$\frac{\Delta/8\sqrt{n}}{12\Delta/\sqrt{n}} \geq \frac{1}{96}$$

This gives us a final success probability of at least $1/200$. ◀

Proof of Lemma 9. As in the previous proof, we will assume that $\pi_v(B)$ and $\pi_v(B')$ concentrate in intervals of radius $\Delta/16\sqrt{n}$ around $\pi_v(z)$ and $\pi_v(z')$ respectively (this happens with probability $1-2\delta$). Now the probability that s intersects $\pi_v(B)$ is $1/96$, since $\pi_v(B)$ is contained in a ball of diameter $\Delta/8\sqrt{n}$ and the partition occurs uniformly in an interval of $12\Delta/\sqrt{n}$. Using Lemma 4, the probability that $\pi_v(B)$ and $\pi_v(B')$ intersect is bounded as follows:

$$\begin{aligned}
& \mathsf{Pr}\left(\|\pi_v(\pi_W(z-z'))\| \le \frac{\Delta}{8\sqrt{n}}\right) \\
\le\ & \mathsf{Pr}\left(\|\pi_v(\pi_W(z-z'))\| \le \left(\frac{\Delta}{8\sqrt{n}}\frac{\sqrt{n-k}}{\|\pi_W(z-z')\|}\right)\frac{\|\pi_W(z-z)\|}{\sqrt{n-k}}\right) \\
\le\ & \sqrt{\frac{2}{\pi}}\frac{\Delta}{8\sqrt{n}}\frac{\sqrt{n-k}}{\|\pi_W(z-z')\|} \\
\le\ & \sqrt{\frac{2}{\pi}}\frac{1}{8}\sqrt{\frac{8}{9}}\frac{\Delta}{\sqrt{9/10}\|z-z'\|} \\
\le\ & \frac{1}{18}\sqrt{\frac{10}{\pi}}\frac{\Delta}{(\Delta/2)-r} \\
\le\ & 0.2.
\end{aligned}$$

The probability of a bad split is upper bounded by

$$2\delta + \mathsf{Pr}\left(\pi_v(B)\cap\pi_v(B')\neq\emptyset\right)\mathsf{Pr}\left(s\in\pi_v(B)\right) < \frac{4}{e^9} + \frac{0.2}{96} < \frac{1}{300}.$$

◀

References

1 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for near neighbor problem in high dimensions. In *STOC*, 2006.

2 Rosa Arriaga and Santosh Vempala. An algorithmic theory of learning: Robust concepts and random projection. *Mach. Learn.*, 63:161–182, May 2006.

3 P. Assouad. Plongements lipschitziens dans r^n. *Bull. Soc. Math. France*, 111:429–448, 1983.

4 Jon Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

5 Alina Beygelzimer, Sham Kakade, and John Langford. Cover tree for nearest neighbor. In *ICML*, 2006.

6 Sanjoy Dasgupta. Hierarchical clustering with performance guarantees. In Hermann Locarek-Junge and Claus Weihs, editors, *Proceedings of the 11th IFCS Biennial Conference and 33rd Annual Conference of the Gesellschaft für Klassifikation e.V., Studies in Classification, Data Analysis, and Knowledge Organization*. Springer-Verlag, 2010.

7 Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *STOC*, 2008.

8 Sanjoy Dasgupta and Yoav Freund. Random projection trees for vector quantization. *IEEE Trans. Inf. Theor.*, 55:3229–3242, July 2009.

9 Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures and Algorithms*, 22(1):60–65, 2003.

10 Aman Dhesi and Purushottam Kar. Random projection trees revisited. In *NIPS*, pages 496–504, 2010.

11 Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS*, pages 534–543, 2003.

12 Sariel Har-Peled. A replacement for voronoi diagrams of near linear size. In *FOCS*, 2001.

13 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

14 Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *STOC*, 1997.

15 William Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

16 Samory Kpotufe. Escaping the curse of dimensionality with a tree-based regressor. In *Conference on Computational Learning Theory*, 2009.

17 Robert Krauthgamer and James Lee. Navigating nets: simple algorithms for proximity search. In *SODA*, 2004.

18 Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 30(2):457–474, 2000.

19 Robert Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

20 Santosh S. Vempala. *The Random Projection Method*. The American Mathematical Society, 2004.

Lower Bounds for the Average and Smoothed Number of Pareto Optima

Navin Goyal[1] **and Luis Rademacher**[2]

1 **Microsoft Research India**
navingo@microsoft.com
2 **Computer Science and Engineering**
Ohio State University
lrademac@cse.ohio-state.edu

Abstract

Smoothed analysis of multiobjective 0–1 linear optimization has drawn considerable attention recently. In this literature, the number of Pareto-optimal solutions (i.e., solutions with the property that no other solution is at least as good in all the coordinates and better in at least one) for multiobjective optimization problems is the central object of study. In this paper, we prove several lower bounds for the expected number of Pareto optima. Our basic result is a lower bound of $\Omega_d(n^{d-1})$ for optimization problems with d objectives and n variables under fairly general conditions on the distributions of the linear objectives. Our proof relates the problem of lower bounding the number of Pareto optima to results in discrete geometry and geometric probability connected to arrangements of hyperplanes. We use our basic result to derive (1) To our knowledge, the first lower bound for natural multiobjective optimization problems. We illustrate this for the maximum spanning tree problem with randomly chosen edge weights. Our technique is sufficiently flexible to yield such lower bounds for other standard objective functions studied in this setting (such as multiobjective shortest path, TSP tour, matching). (2) Smoothed lower bound of $\min\{\Omega_d(n^{d-1.5}\phi^{(d-\log d)(1-\Theta(1/\phi))}), 2^{\Theta(n)}\}$ for the 0–1 knapsack problem with d profits for ϕ-semirandom distributions for a version of the knapsack problem. This improves the recent lower bound of Brunsch and Röglin.

1998 ACM Subject Classification F.2.2 Geometrical problems and computations

Keywords and phrases geometric probability, smoothed analysis, multiobjective optimization, Pareto optimal, knapsack

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.58

1 Introduction

Multiobjective optimization involves scenarios where there is more than one objective function to optimize: When planning a train trip we may want to choose connections that minimize fare, total time, number of train changes, etc. The objectives may conflict with each other and there may not be a single best solution to the problem. Such multiobjective optimization problems arise in diverse fields ranging from economics to computer science, and have been well-studied. A number of approaches exist in the literature to deal with the trade-offs among the objectives in such situations: Goal programming, multiobjective approximation algorithms, Pareto-optimality; see, e.g., [12, 13, 17] for references. It is the latter approach using Pareto-optimality that concerns us in this paper. A Pareto-optimal solution is a solution with the property that no other solution is at least as good in all the objectives and better in at least one. Clearly, the set of Pareto-optimal solutions (Pareto set in short)

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 58–69

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

contains all desirable solutions as any other solution is strictly worse than a solution in the Pareto set. In the worst case, the Pareto set can be exponentially large as a function of the input size (see, e.g., [14]). However, in many cases of interest, the Pareto set is typically not too large. Thus, if the Pareto set is small and can be generated efficiently, then it can be treated possibly with human assistance to choose among the few alternatives. Pareto sets are also used in heuristics for optimization problems (e.g. [15]). To explain why Pareto sets are frequently small in practice, multiobjective optimization has recently been studied from the view-point of smoothed analysis [20]. We introduce some notation before describing this work.

Notation.

For positive integer n, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. The multiobjective optimization problems we study have binary variables and linear objective functions. In a general setting, the feasible solution set is an arbitrary set $\mathcal{S} \subseteq \{0,1\}^n$. The problem has d linear objective functions $v^{(i)} : \mathcal{S} \to \mathbb{R}$, given by $v^{(i)}(x) = \sum_{j\in[n]} v_j^{(i)} x_j$, for $i \in [d]$, and $(v_1^{(i)}, \ldots, v_n^{(i)}) \in \mathbb{R}^n$ (so $v^{(i)}$ is also interpreted as an n-dimensional vector in the natural way). For convenience, we will assume, unless otherwise specified, that we want to maximize all the objectives, and we will refer to the objectives as profits. This entails no loss of generality. Thus the optimization problem is the following.

$$\begin{aligned} &\text{maximize } v^{(1)}(x), \ldots, \text{maximize } v^{(d)}(x), \\ &\text{subject to: } x \in \mathcal{S}. \end{aligned} \tag{1}$$

For the special case of the multiobjective 0–1 knapsack problem we have $d+1$ objectives: One of the objectives will be the weight $w = (w_1, \ldots, w_n)$, which should be minimized, and the other d objectives will be the profits as before: $v^{(i)} = (v_1^{(i)}, \ldots, v_n^{(i)})$ for $i \in [d]$, and all the entries in w and $v^{(i)}$ come from $[0, 1]$.

Let V be the $d \times n$ matrix with rows $v^{(1)}, \ldots, v^{(d)}$. We will use the partial order $\preceq$ in $\mathbb{R}^d$ defined by $x \preceq y$ iff for all $i \in [d]$ we have $x_i \leq y_i$. For $a, b \in \mathbb{R}^d$ we say that b dominates a if $b_i \geq a_i$ for all $i \in [d]$, and for at least one $i \in [d]$, we have strict inequality. We denote the relation of b dominating a by $b \succ a$. A solution $x \in \mathcal{S}$ is said to be Pareto-optimal (or maximal under $\preceq$) if $Vx \not\prec Vy$ for all $y \in \mathcal{S}$. For the knapsack problem, we need to modify the definition of domination appropriately because while we want to maximize the profit, we want to minimize the weight. It will be clear from the context which notion is being used. For a set of points X in Euclidean space, let $p(X)$ denote the number of Pareto-optima in X.

Smoothed analysis.

For our multiobjective optimization problem (1), in the worst case the size of the Pareto set can be exponential even for $d = 2$ (the bicriteria case). Smoothed analysis is a framework for analysis of algorithms introduced by Spielman and Teng [20] to explain the fast running time of the Simplex algorithm in practice, despite having exponential running time in the worst case. Since its introduction, smoothed analysis has been used for a variety of algorithms. Beier and Vöcking [3] studied bicriteria 0–1 knapsack problem under smoothed analysis. In our context of multiobjective optimization, smoothed analysis would mean that the instance (specified by V) is chosen adversarially, but then each entry is independently perturbed according to, say, Gaussian noise with small standard deviation. In fact, Beier and Vöcking [3] introduced a stronger notion of smoothed analysis. In one version of their model, each entry of the matrix V is an independent random variable taking values in $[0, 1]$ with the restriction

that each has probability density function bounded above by ϕ, for a parameter $\phi \geq 1$. We refer to distributions supported on $[-1, 1]$ with probability density bounded above by ϕ as ϕ-semirandom distributions. For more generality, one of the rows of V could be chosen fully adversarially (deterministically). As ϕ is increased, the semirandom model becomes more like the worst case model. With the exception of Theorem 4 below, we do not require adversarial choice of a row in V.

Previous work.

Beier and Vöcking [2] showed that in the above model for the bi-criteria version of the 0–1 knapsack problem with adversarial weights the expected number of Pareto optima is $O(\phi n^4)$. The result generalizes to other bicriteria optimization problems. [1] makes this generalization explicit and improves the upper bound to $O(\phi n^2)$. Röglin and Teng [17] studied the multiobjective optimization problem in the above framework. They showed that the expected size of the Pareto set with d objectives is of the form $O((\phi n)^{2^{d-2}(d+1)!})$. Moitra and O'Donnell [13] improved this upper bound to $2 \cdot (4\phi d)^{d(d+1)/2} \cdot n^{2d}$. (Again, these results allow one of the objectives to be chosen adversarially.) Very recently, this has been improved to $O(n^{2d}\phi^d)$ [6] for the mildly restricted class of quasiconcave ϕ-smooth instances. These papers [17, 13] raised the question of a lower bound on the expected number of Pareto optima.

An early average-case lower bound of $\Omega(n^2)$ was proven in [2] for the knapsack problem with a single profit vector. Their result however required an adversarial choice of exponentially increasing weights. Recently, Brunsch and Röglin [5] proved lower bounds of the form

$$\Omega_d(\min\{(n\phi)^{(d-\log_2 d)\cdot(1-\Theta(1/\phi))}, 2^{\Theta(n)}\}),$$

where Ω_d means that the constant in the asymptotic notation may depend on d. Unfortunately, the instances constructed by them use $\mathcal{S}$ that does not seem to correspond to natural optimization problems.

Our results.

In this paper we prove lower bounds on the expected number of Pareto optima. Our basic result deals with the case when every entry in the matrix V is chosen independently from a distribution with density symmetric around the origin. Note that we do not require that the distributions be identical: Each entry can have a different distribution but we require that the distributions have a density. This generality will in fact be useful in our lower bound for the maximum spanning tree problem, Theorem 2. Note also that all entries of V are random unlike the results discussed above where one of the objectives is chosen adversarially. This makes our lower bound stronger.

▶ **Theorem 1** (Basic theorem). *Suppose that each entry of a $d \times n$ random matrix V is chosen independently according to (not necessarily identical) symmetric distributions with a density. Let X denote the random set $\{Vr : r \in \{0,1\}^n\}$. Then*

$$\mathbb{E}_V\, p(X) \geq \frac{1}{2^{d-1}} \sum_{k=0}^{d-1} \binom{n-1}{k}. \tag{2}$$

This implies the simpler bound $\mathbb{E}_V\, p(X) \geq \left(\frac{n-1}{2(d-1)}\right)^{d-1}$.

We give two proofs of this result. The two proofs have a similar essence, but a somewhat different form. Both proofs relate the problem at hand to some well-known results in geometry.

This connection with geometry is new, and may be useful for future research. The first proof lower bounds the expected number of Pareto-optima of a point set by the expected number of vertices of its convex hull (up to a constant that depends on d but not on n) and then invokes known lower bounds on the expected number of vertices of projections of hypercubes, as the random set in Theorem 1 is a linear image of the vertices of the hypercube. The second proof gives a characterization of maximality in terms of 0–1 vectors and then relaxes integrality to get a relaxed dual characterization by means of convex separation, which reduces the counting of Pareto-optima to lower bounding the probability that the convex hull of n random points contains the origin. This probability is known exactly by a theorem of Wendel (Theorem 6).

Interestingly, our lower bound is basically the same as the expected number of Pareto optima when 2^n uniformly random points are chosen from $[-1,1]^d$, which is shown to be $\Theta_d(n^{d-1})$ in several papers [4, 9, 8]. This raises the possibility of a closer connection between the two models; such a connection could be useful as the model of uniformly random points is better understood.

The basic theorem above corresponds to the case when the set of feasible solutions $\mathcal{S}$ is $\{0,1\}^n$. But in many interesting cases $\mathcal{S}$ is a strict subset of $\{0,1\}^n$: For example, in the multiobjective spanning tree problem n is the number of edges in an underlying network, and $\mathcal{S}$ is the set of incidence vectors of spanning trees in the network; similarly, for the multiobjective shortest path problem $\mathcal{S}$ is the set of incidence vectors of s–t paths. We can use our basic theorem to prove lower bounds on the size of the Pareto set for such $\mathcal{S}$. Our technique is pliable enough to give interesting lower bounds for many standard objective functions used in multiobjective optimization (in fact, any standard objective that we tried): Multiobjective shortest path, TSP tour, matching, arborescence, etc. We will illustrate the idea with the multiobjective spanning tree problem on the complete graph. In this problem, we have the complete undirected graph K_n on n vertices as the underlying graph. Each edge e has a set of profits $v^{(i)}(e) \in [-1,1]$ for $i \in [d]$. The set $\mathcal{S}$ of feasible solutions is given by the incidence vectors of spanning trees of K_n. Notice that the feasible set here lives in $\{0,1\}^{\binom{n}{2}}$ and not in $\{0,1\}^n$.

▶ **Theorem 2.** *In the d objective maximum spanning tree problem on K_n there exists a choice of 4-semirandom distributions such that the expected number of Pareto-optimal spanning trees is at least $(\frac{n-3}{2(d-1)})^{d-1}$.*

The proof of this theorem utilizes the full power of Theorem 1, namely the ability to choose different symmetric distributions.

In our basic theorem above, Theorem 1, we required the distributions to be symmetric, and therefore that theorem does not apply to the 0–1 knapsack problem where all profits and weights are non-negative. With a slight loss in the lower bound we also get a lower bound for this case. In the multiobjective 0–1 knapsack problem we have d objectives $v^{(i)}$ for $i \in [d]$ called profits and an additional objective w called weight. Components of $p^{(i)}$ and w are all chosen from $[0,1]$. We want to maximize the profits and minimize the weight, and so the definitions of domination and Pareto-optimality are accordingly modified.

▶ **Theorem 3.** *For the multiobjective 0–1 knapsack problem where all the weight components are 1 and profit components are chosen uniformly at random from $[0,1]$, the expected number of Pareto optima is $\Omega_d(n^{d-1.5})$.*

Theorems 1 or 3 (depending on whether one wants a bound for non-negative or unrestricted weights and profits) can be used in a simple way as the base case of the argument with

$d+1$ objectives in [5, Section 3] to give the following improved lower bound on the expected number of Pareto optima when the profits are ϕ-semirandom (actually, uniform in carefully chosen intervals of length at least $1/\phi$):

▶ **Theorem 4.** *For any fixed $d \geq 2$ (so that the constants in asymptotic notation may depend on d) and for $n \in \mathbb{N}$ and $\phi > 1$ there exist*

1. *weights $w_1, \ldots, w_n \geq 0$,*
2. *intervals $[a_{ij}, b_{ij}] \subseteq [0,1]$, $i \in [d], j \in [n]$ of length at least $1/\phi$ and with $a_{ij} \geq 0$, and*
3. *a set $\mathcal{S} \subseteq \{0,1\}^n$*

such that if profits $v_j^{(i)}$ are chosen independently and uniformly at random in $[a_{ij}, b_{ij}]$, then the expected number of Pareto-optimal solutions of the $(d+1)$-dimensional knapsack problem with solution set $\mathcal{S}$ is at least

$$\min\{\Omega_d(n^{d-1.5}\phi^{(d-\log d)(1-\Theta(1/\phi))}), 2^{\Theta(n)}\}.$$

For general multiobjective optimization (basically without the restriction of entries being non-negative) the exponent of n becomes exactly d.

The technique of [5] requires $\mathcal{S}$ to be chosen adversarially, and so this is the case for Theorem 4 above as well. To our knowledge, no non-trivial lower bounds were known before our work for natural choices of $\mathcal{S}$. This is addressed by our Theorems 1 ($\mathcal{S} = \{0,1\}^n$) and 2 ($\mathcal{S}$ is the set of spanning trees of the complete graph) above, though these Theorems are for a small constant value of ϕ, and therefore do not clarify what the dependence of ϕ should be.

Recently, Brunsch and Röglin improved the induction step of their lower bound [7]. Combining their improved result with our result yields the lower bound of $\min\{\Omega_d(n^{d-1.5}\phi^d), 2^{\Theta(n)}\}$. This lower bound should be contrasted with the aforementioned upper bounds $2 \cdot (4\phi d)^{d(d+1)/2} \cdot n^{2d}$ [13] and $O(n^{2d}\phi^d)$ [6] (this latter bound is only for the mildly restricted but natural class of quasiconcave ϕ-smooth instances; our lower bound also holds for such quasiconcave perturbations).

2 The basic theorem

In this section we prove Theorem 1. We will include two proofs that, while in essence the same, emphasize the geometric and algebraic views, respectively. Also the second proof is more self-contained. It is perhaps worth mentioning that we first discovered the second proof, and in the course of writing the present paper we found the ideas and known results that could be combined to get a more geometric proof.

2.1 First proof

We will use following result that relates the number of Pareto optima of a point set with the number of vertices of its convex hull:

▶ **Theorem 5** ([4], [16, Theorem 4.7])**.** *Let P be a finite subset of $\mathbb{R}^d$. A vertex of the convex hull of P is maximal under $\preceq$ in at least one of the 2^d assignments of d signs $+$ and $-$ to each of the coordinates of the points of P.*

First proof of Theorem 1. The convex hull of X is a random polytope, a zonotope actually, that is, a linear image of a hypercube or, equivalently, a Minkowski sum of segments. By Theorem 5, every vertex is maximal under our partial order $\preceq$ for at least one of the 2^d reflections involving coordinate hyperplanes. That is

$$|\text{vertices of conv } X| \leq \sum_{\epsilon \in \{-1,1\}^d} p(X \text{ with coordinates of points flipped by signs in } \epsilon) \quad (3)$$

Our symmetry assumption followed by (3) implies[1]

$$\begin{aligned}\mathbb{E}_V(p(X)) &= \frac{1}{2^d}\cdot\sum_{\epsilon\in\{-1,1\}^d}\mathbb{E}\,p(X \text{ with coordinates of points flipped by signs in } \epsilon)\\ &\geq \frac{1}{2^d}\cdot\mathbb{E}\,|\text{vertices of conv } X|\end{aligned}$$

It is known [10, Theorem 1.8] that for V with columns in general position (that is, any d columns are linearly independent, which happens almost surely in our case) the number of vertices is equal to the maximal number of vertices of a d-dimensional zonotope formed as the sum of n segments, and this number is known exactly [11, 31.1.1]. That is, almost surely:

$$|\text{vertices of conv } X| = 2\sum_{k=0}^{d-1}\binom{n-1}{k}.$$

The claimed bound follows. ◀

2.2 Second proof

Some more definitions before getting into the proof: Set $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$ and $\mathbb{R}_- = \{x \in \mathbb{R} : x \leq 0\}$. For $\epsilon \in \{-1,1\}^d$, the orthant associated with ϵ is $\{(\epsilon_1 x_1, \ldots, \epsilon_d x_d) : (x_1,\ldots,x_d) \in \mathbb{R}^d_+\}$. In particular, if ϵ is the all 1's vector then we call its associated orthant the positive orthant, and if ϵ is the all -1's vector then we call its orthant the negative orthant. For a finite set of points $P = \{p_1,\ldots,p_k\} \subseteq \mathbb{R}^d$, the conic hull is denoted $\mathsf{cone}(P) = \{\sum_{i=1}^k \alpha_i p_i : \alpha_i \geq 0\}$ (note that the conic hull is always convex).

We will use the following result by Wendel:

▶ **Theorem 6** ([21], [19, Theorem 8.2.1]). *If $X_1,\ldots,X_n$ are independent random points in $\mathbb{R}^d$ whose distributions are symmetric with respect to 0 and such that with probability 1 all subsets of size d are linearly independent, then*

$$\Pr[0 \notin \text{conv}\{X_1,\ldots,X_n\}] = \frac{1}{2^{n-1}}\sum_{k=0}^{d-1}\binom{n-1}{k}.$$

The linear independence condition holds in particular under the simpler assumption that no hyperplane through the origin is assigned positive probability by any of the n distributions. For example, it holds when the points are i.i.d. at random from the unit sphere.

Second proof of Theorem 1. By linearity of expectation we have

$$\mathbb{E}\,p(X) = \sum_r \Pr[Vr \text{ maximal}].$$

Notice that $\Pr[Vr \text{ maximal}]$ does not depend on r, so we can write
$\mathbb{E}\,p(X) = 2^n \Pr[V1 \text{ maximal}]$.

For the rest of the proof we will focus on finding a lower bound on this last probability. To understand this probability we first rewrite the event $[V1 \text{ maximal}]$ in terms of a different

[1] This idea is from [4]. It is used there in the opposite direction, that is, to get upper bounds on the expected number of vertices from upper bounds on the expected number of maximal points.

event via easy intermediate steps:

$$\begin{aligned}[V1 \text{ maximal}] &= [Vr \not> V1, \ \ \forall r \in \{0,1\}^n] \\ &= [0 \not> V(1-r), \ \ \forall r \in \{0,1\}^n] \\ &= [0 \not> Vr, \ \ \forall r \in \{0,1\}^n].\end{aligned}$$

Now we have $\Pr[0 \not> Vr, \ \ \forall r \in \{0,1\}^n] \geq \Pr[0 \not> Vr, \ \ \forall r \in [0,1]^n]$.

Event $[0 \not> Vr, \ \ \forall r \in [0,1]^n]$ is the same as the event $[\mathsf{cone}(v_1, \ldots, v_n) \cap \mathbb{R}^d_- = \{0\}]$. That is to say, the cone generated by the non-negative linear combinations of $v_1, \ldots, v_n$ does not have a point distinct from the origin that lies in the negative orthant.

By the separability property of convex sets (Hahn-Banach theorem) we have that there exists a hyperplane $H = \{x \in \mathbb{R}^d : \langle u, x \rangle = 0\}$ separating $\mathsf{cone}(v_1, \ldots, v_n)$ and $\mathbb{R}^d_-$. That is, there exists $u \in \mathbb{R}^d_+ \setminus \{0\}$ such that $\mathsf{cone}(v_1, \ldots, v_n) \cdot u \geq 0$ and this implies

$$\Pr[\mathsf{cone}(v_1, \ldots, v_n) \cap \mathbb{R}^d_- = \{0\}] = \Pr[\exists u \in \mathbb{R}^d_+ \setminus \{0\} : \mathsf{cone}(v_1, \ldots, v_n) \cdot u \geq 0].$$

Now

$$\begin{aligned}&\Pr[\mathsf{cone}(v_1, \ldots, v_n) \text{ contained in some halfspace}] \\ &\quad \leq \sum_{\epsilon \in \{-1,1\}^d} \Pr[\mathsf{cone}(v_1, \ldots, v_n) \text{ in a halfspace with inner normal in orthant } \epsilon] \\ &\quad = 2^d \Pr[\exists u \in \mathbb{R}^d_+ \setminus \{0\} : \mathsf{cone}(v_1, \ldots, v_n) \cdot u \geq 0].\end{aligned}$$

Clearly, we have

$$[\mathsf{cone}(v_1, \ldots, v_n) \text{ contained in some halfspace}] = [v_1, \ldots, v_n \text{ contained in some halfspace}].$$

Theorem 6 and the fact that the distribution of v_i is centrally symmetric and assigns measure zero to every hyperplane through 0 imply

$$\begin{aligned}\Pr_V[v_1, \ldots, v_n \text{ contained in some halfspace}] &= \Pr_V[0 \notin \operatorname{conv}\{v_1, \ldots, v_n\}] \\ &= \frac{1}{2^{n-1}} \sum_{k=0}^{d-1} \binom{n-1}{k}.\end{aligned}$$

We conclude:

$$\mathbb{E}\, p(X) \geq \frac{1}{2^{d-1}} \sum_{k=0}^{d-1} \binom{n-1}{k}.$$

◀

The following easy corollary of Theorem 1 will be useful later.

▶ Corollary 7. Theorem 1 holds also when the feasible set $\mathcal{S} = \{-1, 1\}^n$.

▶ Corollary 8. Under the assumptions of Theorem 1 and when the set of feasible solutions is $\mathcal{S} \subseteq \{0,1\}^n$ we have

$$\mathbb{E}_V\, p(X) \geq \frac{|\mathcal{S}|}{2^{n+d-1}} \sum_{k=0}^{d-1} \binom{n-1}{k}. \tag{4}$$

Proof. For any given $r \in \mathcal{S}$, the probability that it is Pareto-optimal in the current instance with solution set restricted to $\mathcal{S}$ is at least the probability that it is Pareto-optimal in the instance with solution set $\{0,1\}^n$. By the symmetry of the distribution this probability is independent of r and by Theorem 1 it is at least

$$\frac{1}{2^{n+d-1}} \sum_{k=0}^{d-1} \binom{n-1}{k}.$$

Linearity of expectation completes the proof. ◀

3 Lower bound for multiobjective maximum spanning trees

In this section we show that our basic result can be used to derive similar lower bounds for $\mathcal{S}$ other than those encountered earlier in this paper. We illustrate this for the case of multiobjective maximum spanning tree problem on the complete graph; for this problem, $\mathcal{S}$ is the set of incidence vectors of all spanning trees on n vertices. The idea of the proof is simple: "Embed" the instance of the basic problem into the instance of the problem at hand. The proof requires the full power of Theorem 1. It is worth noting that the direct use of Cor. 8 does not provide any useful bound for the case of spanning trees. The proof below is easily modified so that all profits are chosen from intervals of non-negative numbers.

We now prove Theorem 2.

Proof. The idea of the proof is to embed an instance of the case $\mathcal{S} = \{0,1\}^{n-2}$ of the basic theorem into the tree instance. We now describe our tree instance. We identify a subgraph G of K_n (the complete graph on n vertices): The vertex set of G is the same as the vertex set of K_n, which for convenience we denote by $\{s, t, u_1, u_2, \ldots, u_{n-2}\}$. The edge set of G consists of the edge (s,t), and edges $(s,u_j), (t,u_j)$. Thus, G consists of $2n-3$ edges. Now we choose the distribution of the profits for each edge of K_n. For edges outside G, the distribution for all profits is identical and it is simply the uniform distribution on $[-1,-1/2]$. For edge (s,t), the distribution is uniform over $[1/2,1]$. And for all other edges it is uniform over $[-1/2,1/2]$. Let $\mathcal{T}$ denote the set of spanning trees which include edge (s,t), and for every other vertex u_j, exactly one of (s,u_j) and (t,u_j). Clearly $|\mathcal{T}| = 2^{n-2}$. The result of the above choices of distributions is that all the Pareto-optimal spanning trees come from $\mathcal{T}$:

▶ Claim 9. For any choices of profits from the intervals as specified above, if a tree T is Pareto-optimal then $T \in \mathcal{T}$.

Proof. Fix any choice of profits as above. Suppose that a tree T' is Pareto-optimal but $T' \notin \mathcal{T}$. Then (1) either T' has an edge e outside $E(G)$, or (2) all its edges are from $E(G)$ but it does not use edge (s,t). In case (1), remove the edges from T' that are not in $E(G)$, and then complete the remaining disconnected graph to a spanning tree using edges from $E(G)$. Clearly, the resulting tree is heavier than T' in each of the d weights. In case (2), add edge (s,t) to T', and from the resulting cycle remove some edge other than (s,t). Again, the resulting tree is heavier than T' in each of the d weights. ◀

In the rest of the proof, i will range over $[d]$. The i'th profit of a spanning tree $T \in \mathcal{T}$, which we will denote by $v^{(i)}(T)$, can be written as follows

$$v^{(i)}(T) = v^{(i)}(st) + \sum_{j=1}^{n-2} (v^{(i)}(s,u_j)x_j + v^{(i)}(t,u_j)(1-x_j)),$$

where $x_j = x_j(T) = 1$ if edge (s, u_j) is in the tree and $x_j = 0$ otherwise. We have

$$(v^{(i)}(s,u_j)x_j + v^{(i)}(t,u_j)(1-x_j)) \\ = \frac{v^{(i)}(s,u_j)+v^{(i)}(t,u_j)}{2} + (v^{(i)}(s,u_j) - v^{(i)}(t,u_j))(x_j - \frac{1}{2}).$$

Now, to compute the lower bound on the expected size of the Pareto set we reveal the v's in two steps: First we reveal $(v^{(i)}(s,u_j)+v^{(i)}(t,u_j))$ for all u_j. Then the conditional distribution of each $(v^{(i)}(s,u_j) - v^{(i)}(t,u_j))$ is symmetric (but can be different for different i). Thus the i'th profit of $T \in \mathcal{T}$ is $v^{(i)}(T) = \sum_{i\in[n-2]}(v^{(i)}(s,u_j) - v^{(i)}(t,u_j))(x_j - 1/2) + A^{(i)}$, where $A^{(i)} = v^{(i)}(s,t) + \sum_{j\in[n-2]} \frac{v^{(i)}(s,u_j)+v^{(i)}(t,u_j)}{2}$. Since $A^{(i)}$ is common to all trees, only the first sum in the profit matters in determining Pareto-optimality. Now we are in the situation dealt with by Cor. 7: For each fixing of $(v^{(i)}(s,u_j) + v^{(i)}(t,u_j))$, we get an instance of Cor. 7, and thus a lower bound of $(\frac{n-3}{2(d-1)})^{d-1}$. Since this holds for each fixing of $(v^{(i)}(s,u_j) + v^{(i)}(t,u_j))$, we get that the same lower bound holds for the expectation without conditioning. ◀

4 0-1 Knapsack

We prove Theorem 3.

Proof. To show our lower bound we will use the obvious one-to-one map between our basic problem with d objectives and the profits of the knapsack problem: Let $v^{(1)}, \ldots, v^{(d)}$ be an instance of our basic problem with all the $v_j^{(i)}$ being chosen uniformly at random from $[-1/2, 1/2]$. Now the profits p are obtained from the v's in the natural way: $p_j^{(i)} = v_j^{(i)} + 1/2$. In general, the set of Pareto optima for these two problems (the basic problem instance and its corresponding knapsack instance) are not the same. We will focus instead on the better behaved set $\mathcal{S} \subseteq \{0,1\}^n$ of solutions having exactly $\lfloor n/2 \rfloor$ ones. From Corollary 8 we get that, in the basic problem restricted to $\mathcal{S}$, the expected number of Pareto optima is at least $\Omega_d(n^{d-1.5})$ (using the well-known approximation $\binom{n}{\lfloor n/2 \rfloor} = \Theta(2^n/\sqrt{n})$).

Now we claim that if $x \in \mathcal{S}$ is Pareto-optimal in the restricted basic problem, then it is also Pareto-optimal in the corresponding (unrestricted) knapsack problem. Let $y \in \{0,1\}^n$ be different from x. There are two cases: If y has more than $\lfloor n/2 \rfloor$ ones, then it cannot dominate x, as y has a strictly higher weight (recall that all the weights are 1). If y has at most $\lfloor n/2 \rfloor$ ones, then enlarge this solution arbitrarily to a solution $y' \succeq y$ with exactly $\lfloor n/2 \rfloor$ ones. The maximality of x implies that y' is worse in some profit, and so is y, as the profits are non-negative. ◀

5 Improved lower bound in the semi-random model

We prove Theorem 4.

Proof. We only describe the differences with the argument in the proof of [5, Theorem 8]. As given by Theorem 3 (but scaling the profits by $1/\phi$, which does not change the set of Pareto-optima), we start with a distribution on knapsack instances with d profits and n_p objects (to be determined later) having unit weights and profits uniformly distributed in $[0, 1/\phi]$, and expected number of Pareto-optima at least $\Omega(n^{d-1.5})$. We use n_q (to be determined later) "cloning steps". Each step introduces d new objects while multiplying the number of Pareto-optima by at least $2^d/d$. As in [5], objects used by the splitting step can have profits that are larger than 1, therefore they are split into many objects with profits

distributed in $[0,1]$, and a suitable choice of the set $\mathcal{S}$ ensures that objects representing the split version of another behave as a group.

A simple modification of the argument leading to [5, Corollary 11], using our base case with n_p objects described in the previous paragraph instead of their base case with 1 object, implies that the expected number of Pareto-optima of the constructed instance is $\Omega(n_p^{d-1.5}(2^d/d)^{n_q})$.

Now we need to choose values of n_p and n_q to get a bound in terms of n. By [5, Lemma 11], the total number of objects is at most

$$n_p + dn_q + \frac{2d^2}{\phi - d}\left(\frac{2\phi}{\phi - d}\right)^{n_q} \tag{5}$$

We choose n_q so that the second term is no more than $n/4$ for n and ϕ sufficiently large. Such a choice of n_q is given by $n_q = \lfloor \hat{n}_q \rfloor$ with

$$\hat{n}_q = \frac{\log \phi}{\log \frac{2\phi}{\phi - d}}$$

when $4d^2 \leq n/4$ and $\phi \geq 2d$.

Clearly there can be no more than 2^n Pareto optima, and therefore there must be a point where increasing ϕ does not increase the lower bound. Say, for

$$\phi \leq \left(\frac{2\phi}{\phi - d}\right)^{n/2d} \tag{6}$$

we have that the second term in (5) is no more than $n/2$. Finally, choosing $n_p = \lfloor n/2 \rfloor$ ensures that (5) is at most n.

As explained in the first paragraph, the expected number of Pareto-optima of the whole construction is at least

$$\Omega\left(n_p^{d-1.5}\left(\frac{2^d}{d}\right)^{n_q}\right) \geq \Omega\left(n_p^{d-1.5}\left(\frac{2^d}{d}\right)^{\hat{n}_q}\right) \geq \Omega(n^{d-1.5}\phi^{(d-\log d)(1-\Theta(1/\phi))}).$$

When ϕ violates (6), we construct the same instance as above with maximum density equal to the unique $\hat{\phi}$ satisfying $\hat{\phi} = \left(\frac{2\hat{\phi}}{\hat{\phi}-d}\right)^{n/2d}$ ($\hat{\phi}$ is about $2^{n/2d}$). We get $\hat{n}^q = n/2d$ and, as before, the expected number of Pareto-optima is at least

$$\Omega\left(n_p^{d-1.5}\left(\frac{2^d}{d}\right)^{n_q}\right) \geq \Omega\left(n^{d-1.5}\left(\frac{2^d}{d}\right)^{n/2d}\right) \geq \Omega(2^{\Theta(n)})$$

◀

6 Discussion and Conclusion

We proved lower bounds for the average and smoothed number of Pareto optima by introducing geometric arguments to this setting. Our lower bound is of the form $\Omega(n^{d-1})$, ignoring the dependence on ϕ. The best upper bound we know, even for $\phi = 1$, is that of Moitra and O'Donnell [13] which is of the form $O(n^{2d-2})$, again ignoring the dependence on ϕ. Thus there is a gap between the upper and lower bounds. As mentioned before, the number of Pareto optima for the case when 2^n points are chosen uniformly at random from $[-1,1]^d$ is $\Theta_d(n^{d-1})$.

Do lower bounds similar to ours hold for any sufficiently large feasible set $\mathcal{S}$? Our techniques can show this for natural objectives, but require arguments tailored to the specific objective. It is desirable to have a general lower bound technique that works for all sufficiently large $\mathcal{S}$. Also, in smoothed lower bounds, to get a good dependence on ϕ we need to use the technique of [5], which requires a very special choice of $\mathcal{S}$. So, a more general question is whether we can prove lower bounds with strong dependence on ϕ for all sufficiently large $\mathcal{S}$.

We now briefly discuss some difficulties in proving lower bounds for general $\mathcal{S}$. One approach to this end is to show a lower bound on the expected size of the Pareto set that depends only on $|\mathcal{S}|$, n and d. Our general technique was to first reduce the problem to lower bounding the expected number of vertices in the projection of the convex hull of the points in $\mathcal{S}$ to a random subspace of dimension d. A special distribution which is instructive to consider here, and also interesting in its own right, is given by the case when we project to a d-dimensional space chosen uniformly at random. The expected number of vertices in the projection has been studied for the special cases of the simplex, the cube, and the cross-polytope (see Schneider [18]). But understanding this number for arbitrary 0/1-polytopes seems difficult. When the subspace to be projected to is of dimension $n-1$, we can write the expected number of vertices in the projection as $C \cdot \sum_{v \in V} a(v)$, where $a(v)$ is the solid angle of the cone polar to the tangent cone at vertex v, and C is a constant depending on n. (Suitable generalizations of this formula are easy to obtain for projection to dimensions smaller than $n-1$, but the case of dimension $n-1$ is sufficient for our purpose here.) This captures the intuitive fact that if the polytope is very pointy at vertex v, then v is more likely to be a vertex in the convex hull. It is natural to ask: Given k, what is the $\mathcal{S} \subseteq \{0,1\}^n$ with $|\mathcal{S}| = k$ that minimizes this expectation? Intuitively, the sum of angles $a(v)$ could be minimized when the vertices are close together, as in a Hamming ball. Note the high-level similarity of the problem at hand to the edge-isoperimetric inequality for the Boolean cube. Unfortunately, our numerical experiments show that this is not the case: Hamming balls are not the minimizers of the expected number of vertices of a random projection.

7 Acknowledgements

We thank Yusu Wang for helpful discussions. We are also very thankful to the Isaac Newton Institute for Mathematical Sciences, Cambridge, UK, where part of this work was carried out during the programme on Discrete Analysis.

References

1 René Beier, Heiko Röglin, and Berthold Vöcking. The smoothed number of pareto optimal solutions in bicriteria integer optimization. In *IPCO*, pages 53–67, 2007.

2 René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. *J. Comput. Syst. Sci.*, 69(3):306–329, 2004.

3 René Beier and Berthold Vöcking. Typical properties of winners and losers in discrete optimization. *SIAM J. Comput.*, 35(4):855–881, 2006.

4 Jon Louis Bentley, H. T. Kung, Mario Schkolnick, and Clark D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, 1978.

5 Tobias Brunsch and Heiko Röglin. Lower bounds for the smoothed number of pareto optimal solutions. In *TAMC*, 2011.

6 Tobias Brunsch and Heiko Röglin. Improved smoothed analysis of multiobjective optimization. In *STOC*, 2012.

7 Tobias Brunsch and Heiko Röglin. Personal communication. June, 2011.

8 Christian Buchta. On the average number of maxima in a set of vectors. *Inf. Process. Lett.*, 33(2):63–65, 1989.

9 Luc Devroye. A note on finding convex hulls via maximal vectors. *Inf. Process. Lett.*, 11(1):53–56, 1980.

10 David Donoho and Jared Tanner. Counting the faces of randomly-projected hypercubes and orthants, with applications. *Discrete & Computational Geometry*, 43:522–541, 2010. 10.1007/s00454-009-9221-z.

11 J.E. Goodman and J. O'Rourke. *Handbook of discrete and computational geometry*. Discrete mathematics and its applications. Chapman & Hall/CRC, 2004.

12 Fabrizio Grandoni, R. Ravi, and Mohit Singh. Iterative rounding for multi-objective optimization problems. In *ESA*, pages 95–106, 2009.

13 Ankur Moitra and Ryan O'Donnell. Pareto optimal solutions for smoothed analysts. In *STOC*, 2011.

14 Matthias Müller-Hannemann and Karsten Weihe. Pareto shortest paths is often feasible in practice. In *Algorithm Engineering*, pages 185–198, 2001.

15 G. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

16 Franco P. Preparata and Michael Ian Shamos. *Computational geometry*. Texts and Monographs in Computer Science. Springer-Verlag, New York, 1985. An introduction.

17 Heiko Röglin and Shang-Hua Teng. Smoothed analysis of multiobjective optimization. In *FOCS*, pages 681–690, 2009.

18 Rolf Schneider. Recent results on random polytopes. *Boll. Un. Mat. Ital.*, Ser. (9), 1:17–39, 2008.

19 Rolf Schneider and Wolfgang Weil. *Stochastic and integral geometry*. Probability and its Applications (New York). Springer-Verlag, Berlin, 2008.

20 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004.

21 J. G. Wendel. A problem in geometric probability. *Math. Scand.*, 11:109–111, 1962.

Exponential Space Improvement for *min-wise* Based Algorithms

Guy Feigenblat[1,2], Ely Porat[1], and Ariel Shiftan[1]

1 **Department of Computer Science**
Bar-Ilan University, Ramat Gan 52900, Israel
{feigeng, porately, shiftaa}@cs.biu.ac.il
2 **IBM Haifa Research Lab, Haifa University Campus**
Mount Carmel, Haifa 31905, Israel

Abstract

In this paper we introduce a general framework that exponentially improves the space, the degree of independence, and the time needed by *min-wise* based algorithms. The authors, in SODA11, [15] introduced an exponential time improvement for *min-wise* based algorithms by defining and constructing an almost *k-min-wise* independent family of hash functions. Here we develop an alternative approach that achieves both exponential time and exponential space improvement. The new approach relaxes the need for approximately *min-wise* hash functions, hence gets around the $\Omega(\log \frac{1}{\epsilon})$ independence lower bound in [23]. This is done by defining and constructing a *d-k-min-wise* independent family of hash functions. Surprisingly, for most cases only 8-wise independence is needed for the additional improvement. Moreover, as the degree of independence is a small constant, our function can be implemented efficiently.

Informally, under this definition, all subsets of size d of any fixed set X have an equal probability to have hash values among the minimal k values in X, where the probability is over the random choice of hash function from the family. This property measures the randomness of the family, as choosing a truly random function, obviously, satisfies the definition for $d = k = |X|$. We define and give an efficient time and space construction of approximately *d-k-min-wise* independent family of hash functions for the case where $d = 2$, as this is sufficient for the additional exponential improvement. We discuss how this construction can be used to improve many *min-wise* based algorithms. To our knowledge such definitions, for hash functions, were never studied and no construction was given before. As an example we show how to apply it for similarity and rarity estimation over data streams. Other *min-wise* based algorithms, can be adjusted in the same way.

1998 ACM Subject Classification F.1.2 Modes of Computation - Online computation

Keywords and phrases Streaming, Min-Wise, Hash Functions, Similarity, On line algorithms, Sub-linear algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.70

1 Introduction

Hash functions are fundamental building blocks of many algorithms. They map values from one domain to another, usually smaller. Although they have been studied for many years, designing hash functions is still a hot topic in modern research. In a perfect world we could use a truly random hash function, one that would be chosen randomly out of all the possible mappings.

Specifically, consider the domain of all hash functions $h : U \to U'$, where $|U| = u$ and $|U'| = u'$. As we need to map each of the u elements in the source into one of the u' possible

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 70–85

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

mappings, the number of bits needed to maintain each function is $u\log(u')$. Since nowadays we often have a massive amount of data to process, this amount of space is not feasible. Nevertheless, most algorithms do not really need such a high level of randomness, and can perform well enough with some relaxations. In such cases one can use a much smaller domain of hash functions. A smaller domain implies lower space requirements at the price of a lower level of randomness.

As an illustrative example, the notion of *2-wise-independent* family of hash functions assures the independence of each pair of elements. It is known that only $2\log(u')$ bits are enough in order to choose and maintain such a function out of the family.

This work is focused on the area of min-hashing. One derivative of min-hashing is *min-wise* independent permutations, which were first introduced in [22, 6]. A family of **permutations** $F \subseteq S_u$ (where S_u is the symmetric group) is **min-wise independent** if for any set $X \subseteq [u]$ (where $[u] = \{0, \ldots, u-1\}$) and any $x \in X$, where π is chosen uniformly at random in F, we have:

$$Pr[\min\{\pi(X)\} = \pi(x)] = \frac{1}{|X|}$$

Similarly, a family of **functions** $\mathcal{H} \in [u] \to [u]$ (where $[u] = \{0, \ldots, u-1\}$) is called **min-wise independent** if for any $X \subseteq [u]$, and for any $x \in X$, where h is chosen uniformly at random in $\mathcal{H}$, we have:

$$Pr_{h\in\mathcal{H}}[\min\{h(X)\} = h(x)] = \frac{1}{|X|}$$

Min hashing is a widely used tool for solving problems in computer science such as estimating similarity [6, 4, 7], rarity [14], transitive closure [8], web page duplicate detection [5, 21, 26, 19], sketching techniques [11, 10], and other data mining problems [18, 13, 1, 3].

One of the key properties of min hashing is that it enables to sample the universe of the elements being hashed. This is because each element, over the random choice of hash function from the family, has equal probability of being mapped to the minimal value, regardless of the number of occurrences of the element. Thus, by maintaining the element with the minimal hash value over the input, one can sample the universe.

Similarity estimation of data sets is a fundamental tool in mining data. It is often calculated using the Jaccard similarity coefficient which is defined by $\frac{|A\cap B|}{|A\cup B|}$, where A and B are two data sets. By maintaining the minimal hash value over two sets of data inputs A and B, the probability of getting the same hash value is exactly $\frac{|A\cap B|}{|A\cup B|}$, which equals the Jaccard similarity coefficient, as described in [6, 4, 7, 8].

Indyk, in [20], was first to give a construction of a small approximately *min-wise* independent family of hash functions, another construction was proposed in [25]. A family of functions $\mathcal{H} \subseteq [u] \to [u]$ is called **approximately min-wise independent**, or ϵ-*min-wise* independent, if, for any $X \subseteq [u]$, and for any $x \in X$, where h is chosen uniformly at random in $\mathcal{H}$, we have:

$$Pr_{h\in\mathcal{H}}[\min\{h(X)\} = h(x)] = \frac{1}{|X|}(1 \pm \epsilon)$$

where $\epsilon \in (0, 1)$ is the desired error bound, and $O(\log(\frac{1}{\epsilon}))$ independence is needed. Pătraşcu and Thorup showed in [23] that $\Omega(\log\frac{1}{\epsilon})$ independence is needed for maintaining an approximately *min-wise* function, hence Indyk's construction is optimal.

Recently, in STOC11 [24], a new novel technique that bypasses the above lower bound was proposed by the same authors. They showed that simple tabulation yields approximately *min-wise* hash function, hence it requires constant time and space for each function.

In a previous paper [15] the authors defined and gave a construction for an **approximately k-min-wise *(*ϵ**-k-min-wise *)* independent** family of hash functions:
A family of functions $\mathcal{H} \subseteq [u] \to [u]$ (where $[u] = \{0 \dots u-1\}$) is called ϵ-k-min-wise independent if for any $X \subseteq [u]$ and for any $Y \subset X$, $|Y| = k$ we have

$$\Pr_{h \in \mathcal{H}} \left[\max_{y \in Y} h(y) < \min_{z \in X - Y} h(z) \right] = \frac{1}{\binom{|X|}{|Y|}} (1 \pm \epsilon),$$

where the function h is chosen uniformly at random from $\mathcal{H}$, and $\epsilon \in (0,1)$ is the error bound. It was also shown in [15] that choosing uniformly at random from a $O(k \log\log \frac{1}{\epsilon} + \log \frac{1}{\epsilon})$-wise independent family of hash functions is approximately *k-min-wise* independence. Formerly, most *min-wise* based applications used k different approximately *min-wise* independent hash functions, i.e. they maintained k different samples. The authors, in [15], proposed to use only one approximately *k-min-wise* independent hash function in order to maintain the k samples, by using the k minimal hash values. As by the definition the k minimal elements are fully independent, the estimators' precision can be preserved. Furthermore, the use of this function was found to reduce exponentially the running time of previous known results for *min-wise* based algorithms. The authors offered a general framework, and gave examples of how to apply it for estimating similarity, rarity and entropy of random graphs. In this paper we take it a step forward and reduce exponentially the space and the degree of independence needed by *min-wise* base algorithms. Recently, Porat and Bachrach in [2] proposed a general technique for constructing fingerprints of massive data streams. The heart of their method lies in using a specific family of pseudo-random hashes shown to be approximately *min-wise* independent, where only one bit is needed to be maintained per function. In other words, one can store just a single bit rather than the full element IDs.

Both approximately *min-wise* and *k-min-wise* hash functions, use low degree of independence (hence potentially low memory and runtime), and therefore are applicable for estimating various metrics in the **data stream models**. In the unbounded data stream model, we consider a stream, in which elements arrive sequentially. Due to the size of the stream, it is only allowed to perform one pass over the data. Furthermore, the storage available is poly-logarithmic in the size of the stream. In the windowed data stream model we consider a predefined size window of size N over the stream, such that all the queries to the stream are related to elements in the current window. Similarly to the unbounded streaming model, we are only allowed one pass over the data and the storage available is poly-logarithmic in the size of the window.

1.1 Our Contribution

In this paper we propose a new approach that 'closes the gap' and reduces exponentially the space, the degree of independence, and the time needed by *min-wise* based algorithms, in addition to the exponential time improvement achieved in SODA11[15]. We do this by defining and constructing a small approximately *d-k-min-wise* independent family of hash functions. As will be discussed here, many *min-wise* based estimators can be adjusted to use our construction, and reduce exponentially the space and the degree of independence consumed.

First, we extend the notion of *min-wise* independent family of hash functions by defining a *d-k-min-wise* independent family of hash functions. Then, we show a construction of an

approximately such family for the practical case where $d = 2$. The construction for general d is a technical generalization of that case, and since $d = 2$ is enough for the improvements, we omit the generalization to the full version. Finally, as a usage example, we show how to apply it to estimating similarity and rarity over data streams.

Under our definition for *d-k-min-wise* hash function, all subsets of size d of any fixed set X have an equal probability to have hash values among the minimal k values in X, where the probability is over the random choice of hash function from the family. The formal definition is given in section 2. The degree of independence and the space needed by our construction, for $d = 2$, is a constant. The lack of dependency on k is surprising, but the intuition behind that is the stability property of the k-th ranked element, for large enough k. Hence, the randomness needed by the function is mainly for the independence of the d elements.

We argue that for most applications it is sufficient to use constant $d = 2$. This yields the need of only 8-wise independent hash functions, which can be implemented efficiently in practice. Our innovative approach gets around the $\Omega(\log \frac{1}{\epsilon})$ lower bound [23] of approximately *min-wise* functions, as our family, by definition, does not have to be approximately *min-wise* independent.

To utilize our construction we propose a simple and general framework for exponential space and degree of independence improvement of *min-wise* based algorithms, such as in [8, 14, 5, 18, 10, 11, 1, 3, 21, 12, 16, 13, 17, 26, 19]. Formerly, *min-wise* based algorithms used either k independent approximately *min-wise* hash functions (which can be implemented using either [20] or [24]), or one approximately *k-min-wise* independent function, as was proposed in [15]. This was done in order to sample k independent elements from the universe, notice that even if we use tabulation [24] we would still need k independent instances of it, hence a multiplicative factor of $O(k)$ in independence, space and time. The *k-min-wise* technique improved exponentially the time needed by *min-wise* based applications. Here we take it a step forward by relaxing the need for k independent samples. We propose to use much less degree of independence, specifically only constant degree, and amplify the precision using probabilistic techniques. In comparison to the technique used by Porat and Bachrach in [2], we use more space (as we maintain the elements' IDs), but our running time is better in more than $O(\log \frac{1}{\epsilon})$ factor.

At a high level, we propose to use several independent approximately *d-k-min-wise* independent functions, where each samples less than k elements (where k is the same as in *k-min-wise*). The elements sampled by each function are d-wise independent, therefore we can use Chebyshev's inequality to bound the precision. Specifically, pair-wise is sufficient for applying Chebyshev, and this is why $d = 2$ should be used. By taking the median out of the independent samples, using Chernoff bound, the precision is amplified. The above procedure does not change the algorithm itself, but only the way it samples, hence it is simple to adapt. We found this to improve exponentially the space and the degree of independence (as it is constant for each function), while maintaining the exponential time improvement in [15].

This approach can be applied in cases where the original estimators values are numeric. In these cases it is possible to take the average and median of the sampled values, hence they can be aggregated (as described above). The estimators' values in most of the applications we considered were indeed numerics, but for the other cases, in which this technique cannot be applied, one can still achieve the exponential time improvement by using *k-min-wise* functions.

As illustrative examples, we propose algorithms which utilize the above framework for similarity and rarity. See table 1.1 for comparison of results.

Table 1.1 Similarity and rarity algorithms comparison in the unbounded data stream model. Time complexity is the expected per item observed, and space is given in words, in upper bounds, for constant failure probability.

	previous [14, 9] result	previous result [15]	this paper
Algorithm's hashing time	$\frac{1}{\epsilon^2}\log\frac{1}{\epsilon}$	$\log^2(\frac{1}{\epsilon^2}\log\log\frac{1}{\epsilon})$	$O(1)$
Additional algorithm's time	$\frac{1}{\epsilon^2}$	$\log\frac{1}{\epsilon^2}$	$O(1)$
Space for storing functions	$\frac{1}{\epsilon^2}\log\frac{1}{\epsilon}$	$\frac{1}{\epsilon^2}\log\log\frac{1}{\epsilon}$	$O(1)$
Space used by algorithm	$\frac{1}{\epsilon^2}$	$\frac{1}{\epsilon^2}$	$\frac{1}{\epsilon^2}$

1.2 Outline

In section 2 we define the notion of *d-k-min-wise* and approximately *d-k-min-wise* independent families. Later, in the first part of section 3, we give an outline and the intuition behind the approximately *d-k-min-wise* construction, which is given in details afterwards. In section 4 we present two usage examples of the framework, then in section 5 we conclude and propose future work. Finally, the appendix contains a lemma needed for the completeness of the construction given in section 3.

2 Definitions and Notations

d-k-min-wise independent family of hash functions, are generalization of *min-wise* and *k-min-wise* independent family of hash functions. Informally, under definition 2.1 below, for any disjoint subsets of the domain X, and Y, where $|X| > k \gg |Y| = d$, the elements of Y have an equal probability to have hash values among the minimal k values in $X \cup Y$. The probability is over the random choice of hash function from the family. In other words, all hash values of Y are less than the $k-d+1$ ranked hash value in X. This property measures the randomness of the family, as choosing a truly random function, obviously, satisfies the definition for $d = k = |X|$.

For the rest of this paper, for any set X we denote $MIN_k(X)$ to be the set of k smallest elements in X, and $RANK_k(X)$ to be the k-th element in X, where the elements are sorted by value. In addition, for any set X, and hash function h we denote $h(X)$ to be the set of all hash values of all elements in X. Finally, we denote $[u]$ to be the universe from which the elements are drawn, we choose $u \gg |X|$. We use these notations to define the following:

▶ **Definition 2.1.** A family of functions $\mathcal{H} \subseteq [u] \to [u]$ (where $[u] = \{0 \dots u-1\}$) is called d-k-min-wise independent if for any $X \subseteq [u]$, $|X| = n - d$, and for any $Y \subseteq [u]$, $|Y| = d$, $X \cap Y = \emptyset$, $d \leq k$, we have

$$\Pr_{h\in\mathcal{H}}[RANK_d(h(Y)) < RANK_{k-d+1}(h(X))] = \frac{\binom{k}{d}}{\binom{n}{d}}$$

Where the function h is chosen uniformly at random from $\mathcal{H}$.

For cases where we allow a small error, the respective definition is:

▶ **Definition 2.2.** A family of functions $\mathcal{H} \subseteq [u] \to [u]$ (where $[u] = \{0 \dots u-1\}$) is called approximately d-k-min-wise independent if for any $X \subseteq [u]$, $|X| = n-d$, and for any $Y \subseteq [u]$,

$|Y| = d$, $X \cap Y = \emptyset$, $d \leq k$, we have

$$\Pr_{h \in \mathcal{H}}[RANK_d(h(Y)) < RANK_{k-d+1}(h(X))] = \frac{\binom{k}{d}}{\binom{n}{d}}(1 \pm \epsilon)$$

Where the function h is chosen uniformly at random from $\mathcal{H}$, and $\epsilon \in (0,1)$ is the error bound.

We will also use the following notation. We let $\Pr[\cdot]$ denote a fully random probability measure over $[u] \to [u]$, and let $\Pr_l[\cdot]$ denote any l-wise independent probability measure over the same domain, for $l \in \mathbb{N}$. Finally, let $t = k - d + 1$ and $m = n - d = |X|$, where k, d and n are drawn from the definitions above.

3 The *d-k-min-wise* Construction

3.1 Construction Outline

In this section we present the outline of the construction, which should give the reader the essence of it. In the subsequent sections we delve into the full technical details.

The main intuition behind the construction is that high enough ranked elements are relatively stable, as opposed to lower ranked elements. As an example, consider the minimal element which is known to be not very stable, and in contrast we show that the probability of a high ranked element to deviate from its expected location decreases rapidly as the deviation increases. By definition our goal is to show that every d elements have almost the same probability to be among the k minimal elements. By utilizing the stability property and using large enough k, we show that the amount of independence needed, i.e. l, is surprisingly only $O(d)$. The relationship between d, k and the amount of independence needed are given in the detailed construction section.

We start by showing that in the fully random case, the probability for the hash values of any d elements to be within the k minimal values, is

$$\Pr[h(y_1), h(y_2), \ldots, h(y_d) < RANK_{k-d+1}(h(X))] = \frac{\binom{k}{d}}{\binom{n}{d}} = \frac{k}{n}\frac{k-1}{n-1}\cdots\frac{k-d+1}{n-d+1}$$

which yields that a totally random function is *d-k-min-wise* independent. We next need to find the amount of independence needed for any l-wise independent ($l \geq 1$) family of hash functions, in order to be close enough (within a multiplicative factor of some $\epsilon \in (0,1)$) to this property. In other words, for the chosen amount of independence, we will show that the difference between the random case and l-wise independent case is $\epsilon\frac{\binom{k}{d}}{\binom{n}{d}}$.

Specifically, we will divide the universe of elements into a set ϕ of non-overlapping blocks b_i, for $i \in \mathbb{Z}$. We construct the blocks s.t. b_0 boundaries are roughly around the expected location of the $(k-d+1)$-th hash value in X. For each block, we will estimate the probability that the $(k-d+1)$-th hash value in X, i.e. $RANK_{k-d+1}(h(X))$, falls within this block's boundaries. Based on the complete probability formula, the probability $\Pr[h(y_1), h(y_2), \ldots, h(y_d) < RANK_{k-d+1}(h(X))]$ in the l-wise case is

$$\sum_{i \in \phi} \Pr_l[RANK_{k-d+1}(h(X)) \in b_i] \cdot$$

$$\Pr_l[h(y_1), \ldots, h(y_d) \leq RANK_{k-d+1}(h(X)) \mid RANK_{k-d+1}(h(X)) \in b_i]$$

The construction yields a *d-k-min-wise* independent family if we show that the difference between the fully random case and the above is $\epsilon\frac{\binom{k}{d}}{\binom{n}{d}}$.

We aim to find the appropriate values of l and k sufficient to have the probability $\Pr_l[RANK_{k-d+1}(h(X)) \in b_i]$ decrease polynomially (roughly $\frac{1}{|i|^{d+1}}$), for each block b_i. Then, by adding to l another d degrees of independence, we can assume the elements of Y are randomly distributed, which is utilized to bound $\Pr_l[h(y_1), \ldots, h(y_d) \leq RANK_{k-d+1}(h(X))]$. Eventually, we use both in order to bound the difference, and to show that it is within the allowed error.

3.2 Construction in Details

In this section we provide a construction for an approximately *d-k-min-wise* independent family of hash functions. We use the notions defined in section 2, and divide the universe into a set ϕ of non-overlapping blocks, which will be defined in the next section.

▶ **Lemma 3.1.**

$$\Pr[h(y_1), h(y_2), \ldots, h(y_d) < RANK_{k-d+1}(h(X))] = \frac{k}{n}\frac{k-1}{n-1}\cdots\frac{k-d+1}{n-d+1}$$

Proof. Consider n ordered elements divided into two groups — one of size $n-d$, and the other of size d. The number of possible locations of the d elements is $\binom{n}{d}$. There are $\binom{k}{d}$ possible locations in which the d elements are among the k smallest elements. Hence, the probability for the d element to be among the k'th smallest elements is:

$$\Pr[h(y_1), h(y_2), \ldots, h(y_d) < RANK_{k-d+1}(h(X))] = \frac{\binom{k}{d}}{\binom{n}{d}} = \frac{k}{n}\frac{k-1}{n-1}\cdots\frac{k-d+1}{n-d+1}$$

◀

Since the blocks in ϕ are non-overlapping $\sum_{i\in\phi}\Pr_l[RANK_{k-d+1}(h(X)) \in b_i] = 1$, using lemma 3.1 we get

$$\Pr[h(y_1), h(y_2), \ldots, h(y_d) < RANK_{k-d+1}(h(X))] =$$

$$\frac{k}{n}\frac{k-1}{n-1}\cdots\frac{k-d+1}{n-d+1}\sum_{i\in\phi}\Pr_l[RANK_{k-d+1}(h(X)) \in b_i]$$

▶ **Lemma 3.2.** *Let d, k, ϵ, n, and $h : [u] \to [u]$ be as in definition 2.2, and denote* $\Delta =$

$$\sum_{i\in\phi}\Pr_l[RANK_{k-d+1}(h(X)) \in b_i] \times$$

$$\left[\Pr_l[h(y_1), \ldots, h(y_d) \leq RANK_{k-d+1}(h(X)) \mid RANK_{k-d+1}(h(X)) \in b_i] - \frac{\binom{k}{d}}{\binom{n}{d}}\right]$$

Any family of l-wise independent hash functions is approximately d-k-min-wise independent if

$$-\epsilon\frac{\binom{k}{d}}{\binom{n}{d}} \leq \Delta \leq \epsilon\frac{\binom{k}{d}}{\binom{n}{d}}$$

Proof. Based on the complete probability formula, in the l-wise independent case

$$\Pr_l[h(y_1), h(y_2), \ldots, h(y_d) < RANK_{k-d+1}(h(X))] =$$

$$\sum_{i\in\phi} \Pr_l[RANK_{k-d+1}(h(X)) \in b_i] \cdot$$
$$\Pr_l[h(y_1), h(y_2), \ldots, h(y_d) < RANK_{k-d+1}(h(X)) \mid RANK_{k-d+1}(h(X)) \in b_i]$$

By definition, any family of l-wise independent is approximately *d-k-min-wise* independent if

$$\Pr_l[h(y_1), h(y_2), \ldots, h(y_d) < RANK_{k-d+1}(h(X))] = \frac{\binom{k}{d}}{\binom{n}{d}}(1 \pm \epsilon)$$

which is satisfied if $-\epsilon\frac{\binom{k}{d}}{\binom{n}{d}} \leq \Delta \leq \epsilon\frac{\binom{k}{d}}{\binom{n}{d}}$

◀

3.3 Blocks partitioning

We divide the universe $[0, |U|]$ into non-overlapping blocks. We construct the blocks around $\frac{t|U|}{m}$ as follows: for $i \in \mathbb{Z}$, $b_i = \left[(1+\epsilon(i-1))\frac{t|U|}{m}, (1+\epsilon i)\frac{t|U|}{m}\right)$, e.g.
$\ldots, b_{-1} = \left[(1-2\epsilon)\frac{t|U|}{m}, (1-\epsilon)\frac{t|U|}{m}\right)$, $b_0 = \left[(1-\epsilon)\frac{t|U|}{m}, \frac{t|U|}{m}\right)$, $b_1 = \left[\frac{t|U|}{m}, (1+\epsilon)\frac{t|U|}{m}\right), \ldots$

Notice that, by the blocks partitioning and according to definition 2.2, the expected number of hash values in X, below the upper boundary of block b_0 is t. This will be utilized for estimating the probability of any d elements in Y to be within the smallest k elements in $X \cup Y$ (below the t-th ranked element in X).
We refer to blocks b_i for $i > 0$ as 'positive blocks' and 'negative blocks' otherwise ($i \leq 0$). For the rest of the paper, we ignore blocks which are outside $[0, |U|]$.

3.4 Bounding $\Pr_l[RANK_t(h(X)) \in b_i]$

We now bound the probability that the t-th ranked element's hash value in X falls into block b_i. We show that the probability is decreasing polynomially with the growth of $|i|$. For convenience, we separate the bound for the negative and positive blocks, and specifically we use 1 as an upper bound for the probabilities of b_0, b_1. In addition, we bound $\Pr_l[RANK_t(h(X)) \in b_i]$, for $i > 1$, with the probability of the t-th ranked element's hash value falling within any block greater than i, i.e. $\Pr_l\left[\cup_{j=i}^{\infty} RANK_t(h(X)) \in b_j\right]$. For $i < 0$ we do a similar procedure by bounding it with $\Pr_l\left[\cup_{j=-\infty}^{i} RANK_t(h(X)) \in b_j\right]$.

▶ **Lemma 3.3.** *For $i > 1$, $d = 2$, $\epsilon \in (0,1)$, $k > d + 2 \cdot 8^{\frac{2}{l}} \frac{(6l)^{1+\frac{1}{l}}}{\epsilon^2} - 1$ and $l = 2d + 2$:*

$$\Pr_l[RANK_t(h(X)) \in b_i] \leq \frac{1}{(i-1)^{d+1}}$$

Proof. For block b_i, $X = \{x_1, \ldots, x_m\}$ we define Z_j to be the following indicator variable

$$Z_j = \begin{cases} 1 & h(x_j) < (1+\epsilon(i-1))\frac{t|U|}{m} \\ 0 & otherwise \end{cases}$$

In addition we define $Z = \sum_j Z_j$, and E_i to be the expected value of Z. Notice that since Z is a sum of indicator variables $E_i = (1 + \epsilon(i-1))\frac{t}{m}(m) = t(1 + \epsilon(i-1))$.
We use the above definitions to show that

$$\Pr_l [RANK_t(h(X)) \in b_i] \leq$$

$$\Pr_l[\text{number of hash values smaller than the lower boundary of block } b_i < t] =$$

$$\Pr_l[Z < t] \leq \Pr_l[E_i - Z \geq E_i - t] \leq \Pr_l[|E_i - Z| \geq E_i - t] =$$

$$\Pr_l[|Z - E_i| \geq t(1 + \epsilon(i-1)) - t]$$

Using Markov's inequality, as l is even:

$$\Pr_l[|Z - E_i| \geq t\epsilon(i-1)] \leq \frac{E(|Z - E_i|^l)}{[t\epsilon i]^l}$$

We use the following from lemma A.1:

$$E(|Z - E_i|^l) \leq 8(6l)^{\frac{l+1}{2}}(E_i)^{\frac{l}{2}}$$

Thus,

$$\Pr_l [RANK_t(h(X)) \in b_i] \leq \frac{8(6l)^{\frac{l+1}{2}}(t(1 + \epsilon(i-1)))^{\frac{l}{2}}}{[t\epsilon(i-1)]^l} = \frac{8(6l)^{\frac{l+1}{2}}(1 + \epsilon(i-1))^{\frac{l}{2}}}{t^{\frac{l}{2}}[\epsilon(i-1)]^l}$$

Note that $t > 2 \cdot 8^{\frac{2}{l}} \frac{(6l)^{1+\frac{1}{l}}}{\epsilon^2}$ for $t = k - d + 1$, and proceed as follows:

$$< \frac{8(6l)^{\frac{l+1}{2}}(1 + \epsilon(i-1))^{\frac{l}{2}}}{[2 \cdot 8^{\frac{2}{l}} \frac{(6l)^{1+\frac{1}{l}}}{\epsilon^2}]^{\frac{l}{2}}[\epsilon(i-1)]^l} = \frac{8(6l)^{\frac{l+1}{2}}(1 + \epsilon(i-1))^{\frac{l}{2}}}{2^{\frac{l}{2}} \cdot 8 \cdot \frac{(6l)^{\frac{l+1}{2}}}{\epsilon^l}[\epsilon(i-1)]^l} = \frac{(1 + \epsilon(i-1))^{\frac{l}{2}}}{2^{\frac{l}{2}}(i-1)^l}$$

$$= \frac{(1 + \epsilon(i-1))^{\frac{l}{2}}}{(2(i-1)^2)^{\frac{l}{2}}} = (\frac{1}{2(i-1)^2} + \frac{\epsilon}{2(i-1)})^{\frac{l}{2}} \leq (\frac{1}{i-1})^{\frac{l}{2}}$$

As $l = 2d + 2$ is defined,

$$\leq (\frac{1}{i-1})^{d+1}$$

◀

The proof for the matching lemma for negative blocks is similar and thus, due to lack of space, is omitted to the full version of the paper.

▶ **Lemma 3.4.** *For $i > 0$, $d = 2$, $\epsilon \in (0, 1)$, $k > d + 2 \cdot 8^{\frac{2}{l}} \frac{(6l)^{1+\frac{1}{l}}}{\epsilon^2} - 1$ and $l = 2d + 2$:*

$$\Pr_l [RANK_t(h(X)) \in b_{-i}] \leq \frac{1}{i^{d+1}}$$

3.5 Bounding Δ

In this section we prove the upper and lower bounds of lemma 3.2, i.e. that $-\epsilon\frac{\binom{k}{d}}{\binom{n}{d}} \leq \Delta \leq \epsilon\frac{\binom{k}{d}}{\binom{n}{d}}$.

▶ **Lemma 3.5.** *For $d = 2$, $\epsilon \in (0,1)$, $k = \Omega(d + \frac{(l)^{1+\frac{1}{l}}}{\epsilon^2})$, $l \geq 2d+2$, and using $(l+d)$ independence:*

$$\Delta \leq \epsilon\frac{\binom{k}{d}}{\binom{n}{d}}$$

Proof.

$$\sum_{i=-\infty}^{\infty} \Pr_{l+d}[RANK_t(h(X)) \in b_i] \times$$

$$\left[\Pr_{l+d}[h(y_1), h(y_2), \ldots, h(y_d) \leq RANK_t(h(X)) \mid RANK_t(h(X)) \in b_i] - \frac{\binom{k}{d}}{\binom{n}{d}}\right]$$

Using d degrees of independence (out of $l+d$) for $h(y_1), h(y_2), \ldots, h(y_d)$, we can assume the elements of Y are randomly distributed:

$$\leq \sum_{i=-\infty}^{\infty} \Pr_{l}[RANK_t(h(X)) \in b_i] \left[(\frac{t}{m})^d(1+\epsilon i)^d - \frac{\binom{k}{d}}{\binom{n}{d}}\right]$$

$$\leq \left[\sum_{i=-\infty}^{0} \left(\Pr_{l}\left[\cup_{j=-\infty}^{i} RANK_t(h(X)) \in b_j\right] - \Pr_{l}\left[\cup_{j=-\infty}^{i-1} RANK_t(h(X)) \in b_j\right]\right)\right.$$
$$\left. + \sum_{i=1}^{\infty} \left(\Pr_{l}\left[\cup_{j=i}^{\infty} RANK_t(h(X)) \in b_j\right] - \Pr_{l}\left[\cup_{j=i+1}^{\infty} RANK_t(h(X)) \in b_j\right]\right)\right] \times$$

$$\left[(\frac{t}{m})^d(1+\epsilon i)^d - \frac{\binom{k}{d}}{\binom{n}{d}}\right]$$

By changing the order we get a telescoping sum as follows:

$$= \sum_{i=-\infty}^{-1} \Pr_{l}\left[\cup_{j=-\infty}^{i} RANK_t(h(X)) \in b_j\right] \left[(\frac{t}{m})^d(1+\epsilon i)^d - (\frac{t}{m})^d(1+\epsilon(i+1))^d\right] +$$

$$\Pr_{l}\left[\cup_{j=-\infty}^{0} RANK_t(h(X)) \in b_j\right] \left[(\frac{t}{m})^d - \frac{\binom{k}{d}}{\binom{n}{d}}\right] +$$

$$\Pr_{l}\left[\cup_{j=1}^{\infty} RANK_t(h(X)) \in b_j\right] \left[(\frac{t}{m})^d(1+\epsilon)^d - \frac{\binom{k}{d}}{\binom{n}{d}}\right] +$$

$$\sum_{i=2}^{\infty} \Pr_{l}\left[\cup_{j=i}^{\infty} RANK_t(h(X)) \in b_j\right] \left[(\frac{t}{m})^d(1+\epsilon i)^d - (\frac{t}{m})^d(1+\epsilon(i-1))^d\right] \leq$$

Applying lemmas 3.3 and 3.4, bounding the probabilities of blocks b_0 and b_1 with 1:

$$\sum_{i=-\infty}^{-1} \frac{1}{|i|^{d+1}} |(\frac{t}{m})^d (1+\epsilon i)^d - (\frac{t}{m})^d (1+\epsilon(i+1))^d| +$$

$$|(\frac{t}{m})^d - \frac{\binom{k}{d}}{\binom{n}{d}}| + |(\frac{t}{m})^d (1+\epsilon)^d - \frac{\binom{k}{d}}{\binom{n}{d}}| +$$

$$\sum_{i=2}^{\infty} \frac{1}{(i-1)^{d+1}} |(\frac{t}{m})^d (1+\epsilon i)^d - (\frac{t}{m})^d (1+\epsilon(i-1))^d| =$$

We now substitute d (recall that $d = 2$)

$$\sum_{i=1}^{\infty} \frac{1}{|i|^3} |(\frac{k-1}{n-2})^2 (\epsilon^2(2i-1) - 2\epsilon))| +$$

$$|(\frac{k-1}{n-2})^2 - \frac{k}{n}\frac{k-1}{n-1}| + |(\frac{k-1}{n-2})^2 (1+\epsilon)^2 - \frac{k}{n}\frac{k-1}{n-1}| +$$

$$\sum_{i=2}^{\infty} \frac{1}{(i-1)^3} |(\frac{k-1}{n-2})^2 (\epsilon^2(2i-1) + 2\epsilon))| \leq$$

$$2\frac{k}{n}\frac{k-1}{n-1} \sum_{i=1}^{\infty} \frac{1}{|i|^3} |\epsilon^2(2i-1) - 2\epsilon| +$$

$$|(\frac{k-1}{n-2})^2 - \frac{k}{n}\frac{k-1}{n-1}| + |(\frac{k-1}{n-2})^2 (1+\epsilon)^2 - \frac{k}{n}\frac{k-1}{n-1}| +$$

$$2\frac{k}{n}\frac{k-1}{n-1} \sum_{i=2}^{\infty} \frac{1}{(i-1)^3} |\epsilon^2(2i-1) + 2\epsilon| \leq$$

$$8\frac{k}{n}\frac{k-1}{n-1}\epsilon$$

◀

The lemma for the lower bound, i.e. $-\Delta \leq \epsilon \frac{\binom{k}{d}}{\binom{n}{d}}$, is similar and due the lack of space is omitted to the full version of the paper.

▶ **Lemma 3.6.** *For $d = 2$, $\epsilon \in (0,1)$, $k = \Omega(d + \frac{(l)^{1+\frac{1}{l}}}{\epsilon^2})$, $l \geq 2d+2$, and using $(l+d)$ independence:* $-\Delta \leq \epsilon \frac{\binom{k}{d}}{\binom{n}{d}}$.

We conclude with the following theorem:

▶ **Theorem 3.7.** *For $d = 2$, $\epsilon \in (0,1)$, $k = \Omega(d + \frac{(l)^{1+\frac{1}{l}}}{\epsilon^2})$, $l \geq 2d+2$, any 8-wise independent family of hash functions is approximately 2-k-min-wise (ϵ-2-k-min-wise).*

Proof. Applying lemma 3.5 and lemma 3.6 to lemma 3.2 concludes the proof. ◀

4 General Framework for *min-wise* Based Algorithms

In this section we propose a general framework that utilizes the construction of *2-k-min-wise* functions for improving many *min-wise* based algorithms, such as in [8, 14, 5, 18, 10, 11, 1, 3, 21, 12, 16, 13, 17, 26, 19]. As was mentioned before, *min-wise* enables us to sample elements from the universe. In other words, sample such that each element has an equal probability for being sampled while ignoring repetitions. Common practice was to use some k independent approximately *min-wise* hash functions, where each samples one element. Usually k depends on the error bound $\epsilon \in (0,1)$ and the failure probability $\tau \in (0,1)$, among other constraints. The drawback is that, these functions requires super constant degree of independence, which impacts both time and space [20, 23]. Recently, in SODA11 [15], a new sampling method was proposed in which, instead of using and maintaining k different functions, they used only one *k-min-wise* independent function. This was found to improve exponentially the time needed for sampling in various *min-wise* based applications. Here we take it a step forward by relaxing the need for k independent samples. Our technique uses much less degree of independence, specifically only constant degree per function, and by probabilistic techniques amplifies the precision.

We propose a procedure that does not change the algorithm itself, but only the way it samples, hence it is simple to adapt. We found that *2-k-min-wise* independent functions are sufficient in order to preserve the precision. In details, one can use $O(\log \frac{1}{\tau})$ approximately *2-k-min-wise* independent functions, where each samples $\frac{k}{\log \frac{1}{\tau}}$ elements (where k is the same as in *k-min-wise*). For cases where the original *min-wise* based estimator values are numeric, one can use the original *min-wise* based estimator on each sampled elements, and then average the values using Chebyshev's inequality. Notice that Chebyshev's inequality can be applied since the elements are 2-wise independent. Next, the precision is amplified by taking the median out of the $\log \frac{1}{\tau}$ groups, and by using Chernoff bound the precision becomes as desired. This procedure improves exponentially the space and time complexity (as the space for each function is constant). For cases where the original *min-wise* based estimator values are not numeric, and therefore averaging and taking the median is not applicable, the *k-min-wise* technique is still valuable for the exponential time improvement. As illustrative examples, the rest of this section describes algorithms which utilizes the above framework for estimating similarity and rarity.

4.1 *2-k-min-wise* Estimator for Similarity

One of the possible uses of our framework is similarity estimation of two data streams. As was mentioned in the introduction, the problem was studied by [14] and recently in [15]. The use of our construction improves exponentially the space and the degree of independence of current known results. We will now present two algorithms for solving the problem, in the unbounded and in the windowed data stream models. The technique we use is general, and can be utilized to improve many *min-wise* based algorithms since most of them handle the *min-wise* functions similarly.

Based on the *k-min-wise* estimator from [9] (which is also formally defined in [15]) we construct *2-k-min-wise* estimator. Let $h_1(A), h_2(A), ..., h_k(A)$ and $h_1(B), h_2(B), ..., h_k(B)$ be k pair-wise independent min hash values for the sets A and B, respectively, and $h_{1...k}(A)$ be the set containing $h_1(A), h_2(A), ..., h_k(A)$. In addition let $S(A,B)$ be the similarity of the two sets. We estimate the similarity by running the following procedure $\log \tau^{-1}$ times, and

choosing the median value of

$$\hat{S}(A,B) = \frac{\mid h_{1\ldots k}(A) \cap h_{1\ldots k}(B) \cap h_{1\ldots k}(A \cup B) \mid}{k},\ 0 < \epsilon < 1, 0 < \tau < 1, k \geq 2\epsilon^{-2}$$

▶ **Theorem 4.1.** *For an error bound ϵ with success probability at least $1 - \tau$:*

$$\hat{S}(A,B)_{\text{median}} \in \frac{|A \cap B|}{|A \cup B|} \pm \epsilon$$

Proof. Let x_i be an indicator variable which equals 1 if the i-th element in $h_{1\ldots k}(A \cup B)$ exists in $h_{1\ldots k}(A) \cap h_{1\ldots k}(B)$, and 0 otherwise. In addition, we define $X = \sum_{i=1}^{k} x_i$. As the indicator variables are pair-wise independent, we can use Chebyshev's inequality to bound the failure probability of $\hat{S}(A,B)$.

$$\Pr\left(|X - k \cdot S(A,B)| \geq k\epsilon\right) \leq \frac{k \cdot S(A,B) \cdot (1 - S(A,B))}{k^2\epsilon^2} \leq \frac{1}{4k\epsilon^2} \leq \frac{1}{4}$$

We next define z_j to be an indicator variable s.t. $z_j = 1$ if the j-th iteration of $\hat{S}(A,B)$ failed, and 0 otherwise. Respectively $Z = \sum_{j=1}^{\log \frac{1}{\tau}} z_i$. Since the $\log \tau^{-1}$ values of $\hat{S}(A,B)$ are independent, we can apply Chernoff bound to increase the success probability. The following is the overall failure probability of the estimator:

$$\Pr\left(Z > \frac{\log \frac{1}{\tau}}{2}\right) < \left(\frac{e}{2^2}\right)^{\frac{\log \frac{1}{\tau}}{4}} < \left(\frac{2^2}{e}\right)^{\frac{-\log \frac{1}{\tau}}{4}} < \left(\frac{2^2}{e}\right)^{\frac{\log \tau}{4}} < \tau$$

◀

4.2 Unbounded Data Stream Algorithm

We now present a similarity estimation algorithm in the unbounded data stream model. Recall that in this model we consider a stream allowing only one pass over the data, and the storage available is poly-logarithmic in the size of the stream. Our algorithm uses the *2-k-min-wise* estimator, with $k = 2\epsilon^{-2}$, and runs it $\log \frac{1}{\tau}$ times as described above.

For each iteration of the estimator, we randomly choose a function from an approximately *2-k-min-wise* independent family of hash functions, using a family of constant degree polynomials over $GF(n)$ (for prime n). In order to maintain the lowest k hash values of the elements observed in the stream, we use a binary tree of size k for each stream. On element arrival, we calculate its hash value and add it to the tree if the value is smaller than the maximal value currently in the tree, and is not already in the tree. Notice that for large enough stream, the expected time of these operations is constant.

At query time we iterate over the $\log \frac{1}{\tau}$ pairs of trees. Each of the trees has k minimal values of the relevant stream. For each pair we take the set of the k lowest hash values among the $2k$ values, $h_{1\ldots k}(A \cup B)$, and intersect it with the two sets of k values, $h_{1\ldots k}(A)$ and $h_{1\ldots k}(B)$. The result of the iteration is the size of the intersections divided by k. The query returns the median among these results.

The space consumption is composed of $O(k \log \frac{1}{\tau})$ words for the trees and $O(\log \frac{1}{\tau})$ for maintaining the hash functions. The expected running time is $O(\log \frac{1}{\tau})$ per element, and $O(k \log \frac{1}{\tau})$ per query. When comparing this result to [15], notice that table 1.1 is for constant failure probability.

4.3 Windowed Data Stream Algorithm

The similarity estimation procedure in this case is similar to the one given in [15]. The difference is that we run it $\log \frac{1}{\tau}$ times in parallel, and use the *2-k-min-wise* estimator (given above). Notice that the value of k here is less than the value of k in [15] by a factor of $\log \frac{1}{\tau}$.

The expected running time per element, per function, is $O(1)$ in amortized, since the time for hashing is constant. As we run the procedure $\log \frac{1}{\tau}$ times in parallel, we get a total of $O(\log \frac{1}{\tau})$. The space consumption is composed of $O(k \log N)$ words per functions, hence $O(k \log N \log \frac{1}{\tau})$ in total.

4.4 An Improved Rarity Estimator

Similar technique can be applied for rarity estimation. Consider a stream A. $\#\alpha$-rare is defined as the number of elements that appears exactly α times, and #distinct is the number of distinct elements in the stream. The α-rarity of the stream is defined as $R_\alpha = \frac{\#\alpha - rare}{\#\text{distinct}}$.

Datar and Muthukrishnan proposed a rarity estimator in [14], using *min-wise* functions. They maintain the minimal hash value and the frequency of its corresponding element for each of the functions. We denote the frequency of the minimal hash value for the j-th function with freq(j), the rarity is then estimated as follows:

$$\hat{R_\alpha}(A) = \frac{|\ \{j | 1 \leq j \leq k', freq(j) = \alpha\}\ |}{k'},\ \hat{R_\alpha}(A) \in R_\alpha(A) \pm \epsilon$$

For $0 < \epsilon < 1$,$0 < \tau < 1$, and $k' \geq 2\epsilon^{-2} \log \tau^{-1}$ for an error bound ϵ, success probability at least $1 - \tau$ and k'-*min-wise* functions.

Using our construction, one can utilize *2-k-min-wise* functions to perform less iterations and improve the overall complexity. This is done by choosing $\log \frac{1}{\tau}$ independent *2-k-min-wise* hash functions, s.t. $k \geq 2\epsilon^{-2}$. For each of the functions we maintain the k minimal values and apply the same estimator ($\hat{R_\alpha}(A)$). The desired result is the median among these values. The precision is preserved because the k values of each function are pair-wise independent, and the $\log \frac{1}{\tau}$ functions are independent. Hence, we can use both Chebyshev's inequality and Chernoff bound, as described before for similarity in theorem 4.1.

5 Conclusion and Future Work

In this paper we introduced a general framework that exponentially improves the time, space and the degree of independence required by *min-wise* based algorithms. This exponential improvements are obtained by defining and constructing *d-k-min-wise* wise hash functions, in which surprisingly for the practical case, where $d = 2$, only 8-wise independent is required. Our approach gets around the $\Omega(\log \frac{1}{\epsilon})$ independence lower bound in [23], as it relaxes the need of approximately *min-wise* functions. Moreover, it does not change the algorithm itself, but only the way it samples, hence it is simple to adapt.

There are few interesting directions for future work. First, to try to nail down the 8 independence required, or alternatively to find a matching lower bound for the approximately *d-k-min-wise* functions. In addition, it would be interesting to try to close the space gap for non-numeric estimators, where one can not utilize our framework as the probabilistic techniques used are not applicable.

References

1 Yoram Bachrach, Ralf Herbrich, and Ely Porat. Sketching algorithms for approximating rank correlations in collaborative filtering systems. In Jussi Karlgren, Jorma Tarhio, and Heikki Hyyrö, editors, *SPIRE*, volume 5721 of *Lecture Notes in Computer Science*, pages 344–352. Springer, 2009.

2 Yoram Bachrach and Ely Porat. Fast pseudo-random fingerprints. volume abs/1009.5791, 2010.

3 Yoram Bachrach, Ely Porat, and Jeffrey S. Rosenschein. Sketching techniques for collaborative filtering. In *The Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 2016–2021, Pasadena, California, July 2009.

4 Andrei Z. Broder. On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCES97*, pages 21–29. IEEE Computer Society, 1997.

5 Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *COM '00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 1–10, London, UK, 2000. Springer-Verlag.

6 Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, New York, NY, USA, 1998. ACM.

7 Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Selected papers from the sixth international conference on World Wide Web*, pages 1157–1166, Essex, UK, 1997. Elsevier Science Publishers Ltd.

8 Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

9 Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning, 1999.

10 Edith Cohen and Haim Kaplan. Summarizing data using bottom-k sketches. In *PODC*, pages 225–234, 2007.

11 Edith Cohen and Haim Kaplan. Tighter estimation using bottom k sketches. *PVLDB*, 1(1):213–224, 2008.

12 Graham Cormode and S. Muthukrishnan. What's new: finding significant differences in network data streams. *IEEE/ACM Trans. Netw.*, 13(6):1219–1232, 2005.

13 Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 271–280, New York, NY, USA, 2007. ACM.

14 Mayur Datar and S Muthukrishnan. Estimating rarity and similarity over data stream windows. In *In Proceedings of 10th Annual European Symposium on Algorithms, volume 2461 of Lecture Notes in Computer Science*, pages 323–334, 2002.

15 Guy Feigenblat, Ely Porat, and Ariel Shiftan. Exponential time improvement for min-wise based algorithms. In *To appear SODA '11: Proceedings of the 22nd annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2011.

16 Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. Processing set expressions over continuous update streams. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 265–276, New York, NY, USA, 2003. ACM.

17 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 281–291, New York, NY, USA, 2001. ACM.

18 Taher H. Haveliwala, Aristides Gionis, Dan Klein, and Piotr Indyk. Evaluating strategies for similarity search on the web. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 432–442, New York, NY, USA, 2002. ACM.

19 Monika Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291, New York, NY, USA, 2006. ACM.

20 Piotr Indyk. A small approximately min-wise independent family of hash functions. In *Journal of Algorithms*, pages 454–456, 1999.

21 Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 141–150, New York, NY, USA, 2007. ACM.

22 K. Mulmuley. Randomized geometric algorithms and pseudo-random generators. In *SFCS '92: Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 90–100, Washington, DC, USA, 1992. IEEE Computer Society.

23 Mihai Pătraşcu and Mikkel Thorup. On the k-independence required by linear probing and minwise independence. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 715–726, 2010.

24 Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. In *Proc. 43rd ACM Symposium on Theory of Computing (STOC)*, 2011. To appear. See also arXiv:1011.5200.

25 Michael Saks, Aravind Srinivasan, Shiyu Zhou, and David Zuckerman. Low discrepancy sets yield approximate min-wise independent permutation families. In *In Proc. International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 29–32. Springer, 1999.

26 Hui Yang and Jamie Callan. Near-duplicate detection by instance-level constrained clustering. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 421–428, New York, NY, USA, 2006. ACM.

A Appendix

▶ **Lemma A.1.** *Let Z_j be a set of indicator variables, let $Z = \sum_j Z_j$, let E_i be the expected value of Z, and let $l = 6$.*

$$E(|Z - E_i|^l) \leq 8(6l)^{\frac{l+1}{2}}(E_i)^{\frac{l}{2}}$$

Proof. The proof is based on Indyk's lemma 2.2 in [20] for the case where l is constant, in particular for the case where $l = 6$. Here we show how one can reduce an exponent factor. Notice that in the original proof the probability $\Pr|Z - E_i| \geq \epsilon E_i$, is estimated by an upper bound, hence one can remove the previous redundant addition in each addend of the sum:

$$E(|Z - E_i|^l) \leq 2\sum_{j=1}^{\infty}((j^l - (j-1)^l) \cdot 2e^{-\frac{j^2}{2E_i^2}E_i})$$

as $l = 6$ the equality is bounded by

$$\leq 2\sum_{j=1}^{\infty}(6j^{l-1} \cdot 2e^{-\frac{j^2}{2E_i^2}E_i})$$

by continuing the proof as in [20] we get, $E(|Z - E_i|^l) \leq 8(6l)^{\frac{l+1}{2}}(E_i)^{\frac{l}{2}}$

◀

An effective characterization of the alternation hierarchy in two-variable logic

Andreas Krebs[1] **and Howard Straubing**[2]

1 **Wilhelm-Schickard-Institut, Universität Tübingen**
Sand 13, 72076 Tübingen, Germany
mail@krebs-net.de*

2 **Computer Science Department, Boston College**
Chestnut Hill, Massachusetts, USA 02467
straubin@cs.bc.edu†

Abstract

We characterize the languages in the individual levels of the quantifier alternation hierarchy of first-order logic with two variables by identities. This implies decidability of the individual levels. More generally we show that two-sided semidirect products with **J** as the right-hand factor preserve decidability.

1998 ACM Subject Classification F.4.3 Formal Languages (D.3.1)

Keywords and phrases FO_2, Quantifier Alternation, J, Pseudovarities, Identities

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.86

1 Introduction

It has been known for some time (Kamp [6], Immerman and Kozen [5]) that every first-order sentence over the base $<$ defining properties of finite words is equivalent to one containing only three variables. The fragment $FO^2[<]$ of sentences that use only two variables, has been the object of intensive study; Tesson and Thérien [18] give a broad-ranging survey of the many places in which the class of languages definable in this logic arises. Weis and Immerman [21] initiated the study of the hierarchy within $FO^2[<]$ based on alternation of quantifiers. They showed, using model-theoretic techniques, that the hierarchy is infinite, but finite for each fixed alphabet.

In [17], the second author provided an algebraic characterization of the levels of the hierarchy, showing that they correspond to the levels of weakly iterated two-sided semidirect products of the pseudovariety **J** of finite $\mathcal{J}$-trivial monoids. This still left open the problem of *decidability* of the hierarchy: effectively determining from a description of a regular language the lowest level of the hierarchy to which the language belongs. This problem was apparently solved in Almeida-Weil [2], from which explicit identities for the iterated product varieties can be extracted. However, an error in that paper called the correctness of these results into question. Here we show that the given identities do indeed characterize these pseudovarieties. In particular, since it is possible to verify effectively whether a given finite monoid satisfies one of these identities, we obtain an effective procedure for exactly determining the alternation depth of a regular language definable in two-variable logic.

* Research partially supported by DFG/KR 4240/1 "Language Recognition by Algebra and Topology"
† Research partially supported by National Science Foundation Grant CCF-0915065

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 86–98

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

We show more generally that the two-sided semidirect product of a pseudovariety with **J** as the right-hand factor preserves decidability. That is, if we have an effective procedure for determining if a given finite monoid belongs to a pseudovariety **V**, then we have such a procedure for $\mathbf{V} * * \mathbf{J}$.

At several junctures, our proof could have been shortened by appealing to known results about the algebra of finite categories and the topological theory of profinite monoids, which are the principal tools of [2]. For example, Theorem 5 is really just the bonded component theorem of Tilson [20] coupled with Simon's Theorem [15] on $\mathcal{J}$-trivial monoids. Lemma 7 closely mirrors the work of Almeida on the structure of the free profinite $\mathcal{J}$-trivial monoid [1]. In order to keep our argument accessible and self-contained, we have chosen to steer clear of these quite technical results. We do discuss finite categories, but only at the most elementary level. Avoiding profinite techniques forces us to give explicit size bounds, but these are of independent interest in decidability questions.

We give the necessary preliminaries from algebra in Section 2. Section 3 is devoted to our fundamental theorem, a category-based characterization of two-sided semidirect products with **J** as the right-hand factor. We apply this result in Section 4 to obtain explicit identities for the levels of the hierarchy, thus solving the decidability problem. We use these identities in Section 5 to give a new proof of the result of Weis and Immerman that the hierarchy collapses for each fixed input alphabet. Section 6 proves the general decidability-preserving result for block products with **J**.

After we circulated an early draft of this paper, we became aware of a number of related results. Kufleitner and Weil [9], building on earlier work of theirs [8], independently established the decidability of the levels of the alternation hierarchy, using an entirely different algebraic characterization. A proof that $\mathbf{V} * * \mathbf{J}$ is decidable if **V** is appears in the unpublished Ph.D. thesis of Steinberg [16].

2 Preliminaries

While the principal application of our results is in finite model theory, this paper contains no formal logic *per se* and is entirely algebraic in content. The reader should consult [17] and [21] for the definition of $\mathrm{FO}^2[<]$ and the alternation hierarchy within it. For our purposes here, they are to be viewed simply as the language classes corresponding to certain varieties of finite monoids, as discussed below.

2.1 Finite monoids and regular languages

See the book by Pin [11] for a detailed treatment of the matters discussed in this subsection and the next; here we give a brief informal review.

A *monoid* is a set M together with an associative operation for which there is an identity element $1 \in M$. If A is a finite alphabet, then A^* is a monoid with concatenation of words as the multiplication. A^* is the *free monoid* on A: this means that every map $\alpha : A \to M$, where M is a monoid, extends in a unique fashion to a homomorphism from A^* into M.

Apart from free monoids, all the monoids we consider in this paper are finite. If M is a finite monoid, then for every element $m \in M$ there is a unique $e \in \{m^k : k > 1\}$ that is *idempotent, i.e.,* $e^2 = e$. We denote this element m^ω.

If M, N are monoids then we say M *divides* N, and write $M \prec N$, if M is a homomorphic image of a submonoid of N.

We are interested in monoids because of their connection with automata and regular languages: A *congruence* on A^* is an equivalence relation $\sim$ on A^* such that $u_1 \sim u_2$,

$v_1 \sim v_2$, implies $u_1v_1 \sim u_2v_2$. The classes of $\sim$ then form a monoid $M = A^*/\sim$, and the map $u \mapsto [u]_\sim$ sending each word to its congruence class is a homomorphism. If $L \subseteq A^*$, then $\equiv_L$, the *syntactic congruence* of L, is the coarsest congruence for which L is a union of congruence classes. The quotient monoid $A^*/\equiv_L$ is called the *syntactic monoid* of L and is denoted $M(L)$.

We say that a monoid M *recognizes* a language $L \subseteq A^*$ if there is a homomorphism $\alpha : A^* \to M$ and a subset X of M such that $\alpha^{-1}(X) = L$. The following proposition gives the fundamental properties linking automata to finite monoids.

▶ **Proposition 1.** *A language $L \subseteq A^*$ is regular if and only if $M(L)$ is finite. A monoid M recognizes L if and only if $M(L) \prec M$.*

2.2 Varieties and identities

A collection $\mathbf{V}$ of finite monoids closed under finite direct products and division is called a *pseudovariety* of finite monoids. (The prefix 'pseudo' is there because of the restriction to finite products, as the standard use of 'variety' in universal algebra does not carry this restriction.)

Given a pseudovariety $\mathbf{V}$, we consider for each finite alphabet A the set $A^*\mathcal{V}$ of regular languages $L \subseteq A^*$ such that $M(L) \in \mathbf{V}$. We call $\mathcal{V}$ the *variety of languages* corresponding to the pseudovariety $\mathbf{V}$. The correspondence $\mathbf{V} \mapsto \mathcal{V}$ is one-to-one, a consequence of the fact that every pseudovariety is generated by the syntactic monoids it contains. We are interested in this correspondence because of its connection with decidability problems for classes of regular languages: To test whether a given language L belongs to $A^*\mathcal{V}$, we compute its syntactic monoid $M(L)$ and test whether $M(L) \in \mathbf{V}$. Since the multiplication table of the syntactic monoid can be effectively computed from any automaton representation of L, decidability for the classes $A^*\mathcal{V}$ reduces to determining whether a given finite monoid belongs to $\mathbf{V}$.

Let Ξ be the countable alphabet $\{x_1, x_2, \ldots\}$. A *term* over Ξ is built from the letters by concatenation and application of a unary operation $v \mapsto v^\omega$. For example, $(x_1x_2)^\omega x_1$ is a term. We will interpret these terms in finite monoids in the obvious way, by considering a valuation $\psi : \Xi \to M$ and extending it to terms by giving concatenation and the ω operator their usual meaning in M. For this reason, we do not distinguish between $(uv)w$ and $u(vw)$, where u, v and w are themselves terms, nor between terms u^ω and $(u^\omega)^\omega$, as these will be equivalent under every valuation.

An *identity* is a formal equation $u = v$, where u and v are terms. We say that a monoid M *satisfies* the identity, and write $M \models (u = v)$, if u and v are equal under every valuation into M. The family of all finite monoids satisfying a given set of identities is a pseudovariety, and we say that the pseudovariety is *defined* by the set of identities. We must stress that the identities we consider here are very special instances of a much more general class of *pseudoidentities*. Under this broader definition, every pseudovariety is defined by a set of pseudoidentities. See, for instance, Almeida [1]. If a pseudovariety $\mathbf{V}$ is defined by a *finite* set of identities of the form we described, then membership of a given finite monoid M in $\mathbf{V}$ is decidable, since we only need to substitute elements of $\mathbf{V}$ for the variables in the identities in every way possible, and check that equality holds in each case.

We consider four particular pseudovarieties that will be of importance in this paper. (In presenting identities we will relax the formal requirement that all terms are over the alphabet $\{x_1, x_2, \ldots\}$, and use a larger assortment of letters for the variables.)

Ap The pseudovariety **Ap** consists of the *aperiodic* finite monoids, those that contain no

nontrivial groups. It is defined by the identity $x^\omega = xx^\omega$. If A is a finite alphabet, then $M(L) \in \mathbf{Ap}$ if and only if L is definable by a first-order sentence over $<$. In other words, the first-order definable languages form the variety of languages corresponding to **Ap** (McNaughton and Papert [10]).

DA The pseudovariety **DA** is defined by the identity

$$(xyz)^\omega y (xyz)^\omega = (xyz)^\omega.$$

There are many equivalent characterizations of this pseudovariety in terms of other identities, the ideal structure of the monoids, and logic. For us the most important ones are these: First, **DA** is also defined by the identities ([14])

$$(xy)^\omega (yx)^\omega (xy)^\omega = (xy)^\omega,\ x^\omega = xx^\omega.$$

Second, let $e \in M$ be idempotent, and let M_e be the submonoid of M generated by the elements $m \in M$ for which $e \in MmM$. Then $M \in \mathbf{DA}$ if and only if $e = eM_e e$ for all idempotents e of M. Finally, if $L \subseteq A^*$ is a language, then $M(L) \in \mathbf{DA}$ if and only if L is definable in $\mathrm{FO}^2[<]$. In other words, the two-variable definable languages form the variety of languages corresponding to **DA**(Thérien and Wilke, [19]).

J The pseudovariety **J** consists of finite monoids that satisfy the pair of identities

$$(xy)^\omega = (yx)^\omega,\ x^\omega = xx^\omega.$$

This is equivalent to the identities

$$(xy)^\omega x = y(xy)^\omega = (xy)^\omega.$$

Alternatively, **J** consists of finite monoids M such that for all $s, t \in M$, $MsM = MtM$ implies $s = t$. Such monoids are said to be $\mathcal{J}$-trivial.

A theorem due to Imre Simon [15] describes the regular languages whose syntactic monoids are in **J**. Let $w \in A^*$. We say that $v = a_1 \cdots a_k$, where each $a_i \in A$, is a *subword* of w if $w = w_0 a_1 w_1 \cdots a_k w_k$ for some $w_i \in A^*$. We define an equivalence relation $\sim_k$ on A^* that identifies two words if and only if they contain the same subwords of length no more than k. In particular, $w_1 \sim_1 w_2$ if and only if w_1 and w_2 contain the same set of letters. Simon's theorem is:

▶ **Theorem 2.** *Let $\phi : A^* \to M$ be a homomorphism onto a finite monoid. Then the following are equivalent:*

- *$M \in \mathbf{J}$.*
- *There exists $k \geq 1$ such that if $w \sim_k w'$, then $\phi(w) = \phi(w')$. (In particular, M is a quotient of $A^*/\sim_k$.)*

It is easy to show that the second condition implies the first; the deep content of the theorem is the converse implication. The theorem can also be formulated in first-order logic: The variety of languages corresponding to **J** consists of languages definable by Boolean combinations of Σ_1 sentences over $<$.

$\mathbf{J_1}$ The pseudovariety $\mathbf{J_1}$ consists of all idempotent and commutative monoids; *i.e.,* those finite monoids that satisfy the identities $x^2 = x,\ xy = yx$. A language $L \subseteq A^*$ is in the variety of languages corresponding to $\mathbf{J_1}$ if and only if it is a union of $\sim_1$-classes. It is well known, and easy to show, that $\mathbf{J_1} \subseteq \mathbf{J} \subseteq \mathbf{DA} \subseteq \mathbf{Ap}$, and all the inclusions are proper.

2.3 Two-sided Semidirect Products

In this section we describe an operation on pseudovarieties of finite monoids, the *two-sided semidirect product.* This was given in its formal description by Rhodes and Tilson [12], but it has precursors in automata theory in the work of Schützenberger on sequential bimachines [13], Krohn, Mateosian and Rhodes [7], and Eilenberg on triple products [4]. Traditionally, one begins with a two-sided semidirect product operation on monoids, and then uses this to define the corresponding operation on pseudovarieties. Here we find it simpler to define the operation on varieties directly.

Let A be a finite alphabet, and $\psi : A^* \to N$ a homomorphism into a finite monoid. Let $\Sigma = N \times A \times N$, which we treat as a new finite alphabet. We define a length-preserving transduction (not a homomorphism) $\tau_\psi : A^* \to \Sigma^*$ by $\tau_\psi(1) = 1$, and

$$\tau_\psi(a_1 \cdots a_n) = \sigma_1 \cdots \sigma_n, \text{ where}$$

$$\sigma_i = (\psi(a_1 \cdots a_{i-1}), a_i, \psi(a_{i+1} \cdots a_n)) \in \Sigma.$$

(If $i = 1$, we interpret the right-hand side as $(1, a_1, \psi(a_2 \cdots a_n))$, and similarly if $i = n$.)

Let $\mathbf{V}$ and $\mathbf{W}$ be pseudovarieties of finite monoids. Let M be a finite monoid, and let $\phi : A^* \to M$ be a surjective homomorphism. We say that $M \in \mathbf{V} * \mathbf{W}$ if and only if there exist homomorphisms

$$\psi : A^* \to N \in \mathbf{W},$$

$$h : (N \times A \times N)^* \to K \in \mathbf{V},$$

such that ϕ factors through $(h \circ \tau_\psi, \psi)$—in other words, for all $v, w \in A^*$, if $\psi(v) = \psi(w)$ and $h(\tau_\psi(v)) = h(\tau_\psi(w))$, then $\phi(v) = \phi(w)$. It is not difficult to check that this is independent of the alphabet A and the homomorphism ϕ, and is thus determined entirely by M, and that furthermore $\mathbf{V} * \mathbf{W}$ forms a pseudovariety of finite monoids. We will treat this as the definition of $\mathbf{V} * \mathbf{W}$, but it is also straightforward to verify that this coincides with the pseudovariety generated by two-sided semidirect products $K * N$, where $K \in \mathbf{V}$ and $N \in \mathbf{W}$.

We define a sequence $\{\mathbf{V}_i\}_{i \geq 1}$ of pseudovarieties by setting $\mathbf{V}_1 = \mathbf{J}$, and, for $i \geq 1$, $\mathbf{V}_{i+1} = \mathbf{V}_i * \mathbf{J}$. The main result of [17] is that $\mathbf{DA}$ is the union of the pseudovarieties $\mathbf{V}_i$, and that the variety of languages corresponding to $\mathbf{V}_i$ is the i^{th} level of the alternation hierarchy within $\mathrm{FO}^2[<]$.

2.4 Finite categories

We give a brief account of the tools from the algebraic theory of finite categories needed to prove our main results. The original papers of Tilson [20] and Rhodes and Tilson [12] give a complete and careful exposition of the general theory.

The categories studied in category theory are typically big categories, in which the object class consists of something like all topological spaces, and the arrows are all continuous functions. The work of Tilson [20] showed the utility of studying very small categories in which the object set, as well as each set of arrows between two objects, is finite.

A category $\mathcal{C}$ consists of a set of *objects* $\mathrm{obj}(\mathcal{C})$, a set of *arrows* $\hom(A, B)$ from A to B for all $A, B \in \mathrm{obj}(\mathcal{C})$, and associative partial binary operations $\circ : \hom(A, B) \times \hom(B, C) \to \hom(A, C)$ for all $A, B, C \in \mathrm{obj}(\mathcal{C})$ called *composition*, such that there is an identity in $\hom(A, A)$ for all $A \in \mathrm{obj}(\mathcal{C})$.

In this view, a finite monoid is simply a category with a single object, and a finite category is consequently a generalized finite monoid.

Let A be a finite alphabet, M and N finite monoids with homomorphisms

$$M \xleftarrow{\phi} A^* \xrightarrow{\psi} N,$$

where ϕ maps onto M. We will define a finite category, which we call the *kernel category* $\ker(\psi \circ \phi^{-1})$. [1]

The *objects* of $\ker(\psi \circ \phi^{-1})$ are pairs $(n_1, n_2) \in N \times N$. The *arrows* are represented by triples

$$(n_1, n_2) \xrightarrow{u} (n_1', n_2'),$$

where $u \in A^*$, $n_1' = n_1 \cdot \psi(u)$ and $\psi(u) \cdot n_2' = n_2$. Whenever we have a pair of consecutive arrows

$$(n_1, n_2) \xrightarrow{u} (n_1', n_2'), (n_1', n_2') \xrightarrow{v} (n_1'', n_2''),$$

then we can define the product arrow

$$(n_1, n_2) \xrightarrow{uv} (n_1'', n_2'').$$

If this were all there were to arrows in the kernel category, we would in general have an infinite set of arrows between two objects. However, we identify two coterminal arrows

$$(n_1, n_2) \xrightarrow{u,u'} (n_1', n_2')$$

if for all $v, w \in A^*$ with $\psi(v) = n_1$, $\psi(w) = n_2'$,

$$\phi(vuw) = \phi(vu'w).$$

It is easy to check that this identification is compatible with the product on consecutive arrows, so the true arrows of $\ker(\psi \circ \phi^{-1})$ are equivalence classes modulo this identification. In particular, the finiteness of M and N implies that there are only finitely many distinct arrows.

If $(n_1, n_2) = (n_1', n_2')$, then any pair of arrows from (n_1, n_2) to itself are consecutive, and thus the set of all such arrows at (n_1, n_2) is a finite monoid, which we denote M_{n_1,n_2}. This is a *base monoid.* Base monoids, then, are just built from words u satisfying $n_1 \cdot \psi(u) = n_1$, and $\psi(u) \cdot n_2 = n_2$, and collapsing modulo the equivalence relation identifying arrows.

The following lemma concerning the structure of the base monoids will be quite useful.

▶ **Lemma 3.** *Let A be a finite alphabet: M, N, N' finite monoids, and consider homomorphisms*

$$M \xleftarrow{\phi} A^* \xrightarrow{\psi} N \xrightarrow{\psi'} N',$$

where ϕ maps onto M. Then every base monoid of $\ker(\psi \circ \phi^{-1})$ *divides some base monoid of* $\ker((\psi'\psi) \circ \phi^{-1})$.

Proof. Let $n_1, n_2 \in N$. We denote by M_1 the base monoid at (n_1, n_2) in $\ker(\psi \circ \phi^{-1})$, and by M_2 the base monoid at $(n_1', n_2') = (\psi'(n_1), \psi'(n_2))$ in $\ker((\psi'\psi) \circ \phi^{-1})$. Set

$$U = \{u \in A^* : n_1 \cdot \psi(u) = n_1, n_2 = \psi(u) \cdot n_2\},$$

$$U' = \{u \in A^* : n_1' \cdot \psi'\psi(u) = n_1', n_2' = \psi'\psi(u) \cdot n_2'\}.$$

[1] The odd notation for the kernel category is used to maintain consistency with the traditional setting for these finite categories. $\psi \circ \phi^{-1}$ is a *relational morphism* from M to N, and Tilson defines these categories for arbitrary relational morphisms, not just those derived from morphisms of the free monoid.

U and U' are submonoids of A^*, and $U \subseteq U'$. M_1 and M_2 are the quotients of U and U' by the congruences identifying equivalent arrows in the respective categories. Let $u, u' \in U$ represent equivalent arrows of M_2, and suppose $v, w \in A^*$ are such that $\psi(v) = n_1$, $\psi(w) = n_2$. Then $\psi'\psi(v) = n_1'$, $\psi'\psi(w) = n_2'$, so by equivalence in M_2 we have $\phi(vuw) = \phi(vu'w)$. But this means that u and u' represent equivalent arrows in M_1, so M_1 is a quotient of the image of U in M_2. Thus $M_1 \prec M_2$. ◀

It is worth keeping in mind the somewhat counterintuitive message of this lemma: The category $\ker(\psi \circ \phi^{-1})$ is *bigger* (it has more objects) than $\ker((\psi'\psi) \circ \phi^{-1})$ but its base monoids are *smaller*.

The reason for the construction of the kernel category in [12] is its relation to two-sided semidirect products. Roughly speaking, $M \in \mathbf{V} * * \mathbf{W}$ if and only if there exists $\psi : A^* \to N \in \mathbf{W}$ such that the category $\ker(\psi \circ \phi^{-1})$ is 'globally in $\mathbf{V}$'. We will not define this precisely, but instead note without proof one consequence, namely that if $M \in \mathbf{V} * * \mathbf{W}$, then $\ker(\psi \circ \phi^{-1})$ satisfies a weaker condition of being 'locally in $\mathbf{V}$':

▶ **Proposition 4.** *Let $\phi : A^* \to M$ be a homomorphism mapping onto M. If $M \in \mathbf{V} * * \mathbf{W}$, then there is a homomorphism $\psi : A^* \to N \in \mathbf{W}$ such that each base monoid of $\ker(\psi \circ \phi^{-1})$ is in $\mathbf{V}$.*

3 A local-global theorem for categories

In general, the converse of Proposition 4 is false. This section is devoted to establishing an important instance in which it is true, namely when $\mathbf{W} = \mathbf{J}$.

▶ **Theorem 5.** *Let A be a finite alphabet, M and N finite monoids with $N \in \mathbf{J}$ and homomorphisms*

$$M \xleftarrow{\phi} A^* \xrightarrow{\psi} N,$$

where ϕ maps onto M. Suppose $\mathbf{V}$ is a pseudovariety of finite monoids with $\mathbf{J_1} \subseteq \mathbf{V}$. If every base monoid of $\ker(\psi \circ \phi^{-1})$ is in $\mathbf{V}$, then $M \in \mathbf{V} * * \mathbf{J}$.

Proof. It follows from Theorem 2 that for some $k > 0$, ψ factors through the homomorphism $A^* \to A^*/\sim_k$ identifying two words that have the same subwords up to length k. By Lemma 3 we may assume that ψ *is* this homomorphism, and that $N = A^*/\sim_k$. In particular, if $w \in A^*$, then we can represent $\psi(w)$ as the set of subwords of w of length no more than k.

The set $\mathcal{P}(N \times N)$ of subsets of $N \times N$ forms an idempotent and commutative monoid with union as the operation, and hence belongs to $\mathbf{J_1} \subseteq \mathbf{V}$. Let $\Sigma = N \times A \times N$ and let $h_U : \Sigma^* \to \mathcal{P}(N \times N)$ be the homomorphism defined by mapping $\sigma = (P, a, S)$ to $\{(P\psi(a), S)\}$ for each $\sigma \in \Sigma$. Given $P, S \in N$, define a homomorphism $h_{P,S} : \Sigma^* \to M_{P,S}$ by mapping $(P', a, S') \in \Sigma$ to the arrow class of $(P, S) \xrightarrow{a} (P, S)$ if $P = P' = P\psi(a), \psi(a)S = S' = S$, and to $1 \in M_{P,S}$ otherwise. Finally, set M' to be the direct product

$$M' = \mathcal{P}(N \times N) \times \prod_{(P,S) \in N \times N} M_{P,S},$$

and set

$$h = h_U \times \prod_{(P,S) \in N \times N} h_{P,S}.$$

By our hypothesis $M' \in \mathbf{V}$.

Let $w, w' \in A^*$, with $\psi(w) = \psi(w')$ and $h(\tau_\psi(w)) = h(\tau_\psi(w'))$. We will show $\phi(w) = \phi(w')$, which gives the result.

We will look at the paths through $\ker(\psi \circ \phi^{-1})$ traced out by w and w'. Since $\psi(w) = \psi(w')$, the two paths are coterminal, beginning at the object $(1, \psi(w))$ and ending at $(\psi(w), 1)$. Let the i^{th} letter of w be a_i, then the i^{th} arrow on this path is the class of

$$(P_{i-1}, S_{i-1}) \xrightarrow{a_i} (P_i, S_i),$$

where P_j is $\psi(u)$ for the prefix $u = a_1 \cdots a_j$ of length j of w, and likewise $S_j = \psi(v)$ for the suffix $v = a_{j+1} \cdots a_{|w|}$. Let

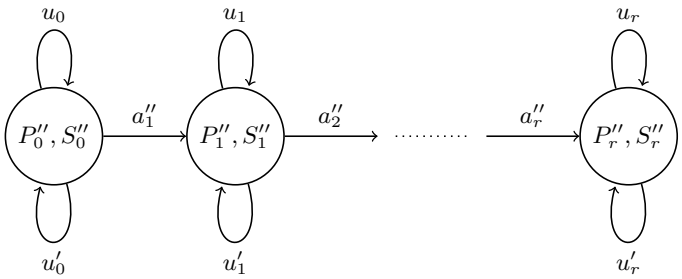
$$(P, S) \xrightarrow{a_i} (P', S')$$

be on the path traced by w, then we have $P \subseteq P'$ and $S' \subseteq S$. Either $P = P'$ and $S = S'$, in which case this arrow belongs to one of the base monoids, or at least one of the inclusions is proper. Since $h_U(\tau_\psi(w)) = h_U(\tau_\psi(w'))$ and $\psi(w) = \psi(w')$, the same pairs $(P, S), (P', S')$ must occur in the path traced by w'. Because of the inclusions, they must occur in the same relative order in this path, with (P, S) preceding (P', S'). They also must be adjacent in this path, since if there were a third pair (P'', S'') between them, we would have

$$P \subseteq P'' \subseteq P', S' \subseteq S'' \subseteq S,$$

so this new pair would have to occur in the original path traced by w, strictly between (P, S) and (P', S'). Finally, the letter a labeling the arrow joining these two objects in the respective paths is completely determined by (P, S) and (P', S'). This is because at least one of the two inclusions $P \subseteq P'$ and $S' \subseteq S$ is proper. Assume without loss of generality that the first of these is a proper inclusion. Then P' contains a word that is not in P, and the last letter of this word is a.

Thus our two paths are depicted by the diagram below:



The paths traverse exactly the same sequence of distinct objects

$$(1, \psi(w)) = (P''_0, S''_0), (P''_1, S''_1), \ldots, (P''_r, S''_r) = (\psi(w), 1).$$

The arrow joining (P''_{j-1}, S''_{j-1}) and (P''_j, S''_j) in both these paths is the same letter a''_j. For $j = 0, \ldots, r$ each path contains a loop at (P''_j, S''_j) labeled by a factor u_j of w in one path, and a factor u'_j in the other path. We have

$$w = u_0 a''_1 u_1 \cdots a''_r u_r,$$

$$w' = u'_0 a''_1 u'_1 \cdots a''_r u'_r.$$

Let $w_0 = w$, $w_{r+1} = w'$ and for $j = 1, \ldots, r$, let

$$w_j = u'_0 a''_1 \cdots u'_{j-1} a''_j u_j \cdots a''_r u_r.$$

In other words, we transform w into w' one step at a time, changing each u_j in succession to u'_j. We claim that at each step, $\phi(w_j) = \phi(w_{j+1})$, so that we will get $\phi(w) = \phi(w')$, as required. Let $(P, S) = (P''_j, S''_j)$. By the definition of $h_{P,S}$, the image in the base monoid $M_{P,S}$ of a word under $h_{P,S} \circ \tau_\psi$ involves only the letters where the prefix under ψ maps to P and suffix maps to S; in our case these are the letters of u_j and u'_j, respectively. So since $h(\tau_\psi(w)) = h(\tau_\psi(w'))$, $(P, S) \xrightarrow{u_j, u'_j} (P, S)$ are equivalent arrows. Thus

$$\phi(w_j) = \phi(u'_0 a''_1 \cdots u'_{j-1} a''_j u_j a''_{j+1} \cdots u_r) = \phi(u'_0 a''_1 \cdots u'_{j-1} a''_j u'_j a''_{j+1} \cdots u_r) = \phi(w_{j+1}).$$

◀

As we mentioned in the introduction, our argument is essentially the one used to prove a more general result, due to Tilson [20], that every category divides a direct product of its strongly connected components. In the special case that we consider, these components reduce to single objects and hence are the base monoids of the category. The hypothesis $\mathbf{J_1} \subseteq \mathbf{V}$ in the statement of Theorem 5 is actually not necessary, so long as $\mathbf{V}$ is nontrivial: If $\mathbf{V}$ is a pseudovariety of monoids that does not contain $\mathbf{J_1}$, then every member of $\mathbf{V}$ is a group, and it is known that the converse of Proposition 4 holds when $\mathbf{V}$ contains only nontrivial groups; this also follows from results in [20]. We do not require this fact in our main application to the alternation hierarchy.

4 Effective characterization of levels of the alternation hierarchy

We now define a sequence of identities that will allow us to characterize the varieties $\mathbf{V}_n$. We set

$$u_1 = (x_1 x_2)^\omega, v_1 = (x_2 x_1)^\omega,$$

and for $n \geq 1$,

$$u_{n+1} = (x_1 \cdots x_{2n} x_{2n+1})^\omega u_n (x_{2n+2} x_1 \cdots x_{2n})^\omega,$$
$$v_{n+1} = (x_1 \cdots x_{2n} x_{2n+1})^\omega v_n (x_{2n+2} x_1 \cdots x_{2n})^\omega.$$

▶ **Theorem 6.** *Let $n \geq 1$. $M \in \mathbf{V}_n$ if and only if $M \models (u_n = v_n)$, and $M \models (x^\omega = xx^\omega)$.*

As we remarked above, when a pseudovariety $\mathbf{V}$ is defined by a finite set of identities of this type, one can decide membership in $\mathbf{V}$. Since the levels of the alternation hierarchy in $\mathrm{FO}^2[<]$ are the varieties of languages corresponding to the $\mathbf{V}_i$, the alternation depth of a language in $\mathrm{FO}^2[<]$ is effectively computable.

Proof. The 'only if' part (the identities hold in $\mathbf{V}_n$) is proved in [17], so we will just give the proof of the 'if' part (sufficiency of the identities).

We prove the theorem by induction on n. It is well known that the identities $u_1 = v_1, x^\omega = xx^\omega$ characterize $\mathbf{V}_1 = \mathbf{J}$.

So we assume $n > 1$ and suppose that M is an aperiodic monoid such that $M \models (u_n = v_n)$. We let ϕ, ψ be as in the previous section, so that ϕ is any morphism mapping onto $M \in \mathbf{V}_n$, and ψ depends on the choice of a subword length K. We will show that if K is chosen to be a large enough value, then each base monoid $M_{P,S}$ of the category $\ker(\psi \circ \phi^{-1})$ satisfies the identity $u_{n-1} = v_{n-1}$. By the inductive hypothesis and since $M_{P,S}$ is aperiodic, this implies that each $M_{P,S}$ belongs to $\mathbf{V}_{n-1}$, and thus by Theorem 5, $M \in \mathbf{V}_{n-1} ** \mathbf{J} = \mathbf{V}_n$.

We let $x_1, \ldots, x_{2(n-1)}$ be any elements of $M_{P,S}$. Thus, each x_i is represented by a triple $(P, S) \xrightarrow{w_i} (P, S)$, where $w_i \in A^*$, $P \cdot \psi(w_i) = P$, $\psi(w_i) \cdot S = S$.

We construct words $W_{n-1}, W'_{n-1} \in A^*$ by replacing each x_i in u_{n-1} (respectively v_{n-1}) by w_i. We will think of ω in these strings as representing a finite exponent N such that $x^N = x^{N+1}$ for all $x \in M$, and hence for all $x \in M_{P,S}$, since each base monoid $M_{P,S}$ divides M. Thus if $n > 2$,

$$W_{n-1} = (w_1 w_2 \cdots w_{2n-3})^N W_{n-2} (w_{2n-2} w_1 \cdots w_{2n-4})^N,$$

$$W'_{n-1} = (w_1 w_2 \cdots w_{2n-3})^N W'_{n-2} (w_{2n-2} w_1 \cdots w_{2n-4})^N.$$

In the special case $n = 2$, we have $W_1 = (w_1 w_2)^N$, $W'_1 = (w_2 w_1)^N$.

If $w \in A^*$, we denote by $\alpha(w)$ the set of letters occurring in A^*. We also denote by B the set $\alpha(W_{n-1}) = \alpha(W'_{n-1})$. Let $z, y \in A^*$ with $\psi(z) = P$, $\psi(y) = S$.

▶ **Lemma 7.** *If $K > |M| \cdot (|A|^2 + |A|)/2$, then z has a suffix z' with a factorization $z' = z_1 z_2 \cdots z_{|M|}$ where*

$$B \subseteq \alpha(z_1) = \alpha(z_2) = \cdots = \alpha(z_{|M|}),$$

and, likewise, y has a prefix y' with a factorization $y' = y_1 y_2 \cdots y_{|M|}$ where

$$B \subseteq \alpha(y_1) = \alpha(y_2) = \cdots = \alpha(y_{|M|}).$$

Assuming the lemma, we will now complete the proof of Theorem 6. Since $M \models (u_n = v_n)$, we obtain $M \models (xy)^\omega (yx)^\omega (xy)^\omega = (xy)^\omega$ by setting $x_1 = x$, $x_2 = y$, and $x_k = 1$ for $k > 2$. Thus $M \in \mathbf{DA}$.

We can write $z = z''z'$, where $z' = z_1 \dots z_{|M|}$ has a factorization as in Lemma 7. By the standard pumping argument, it follows that there are indices $i \le j$ such that $\phi(z_1 \dots z_{i-1})\phi(z_i \cdots z_j) = \phi(z_1 \dots z_{i-1})$, and thus

$$\phi(z_1 \dots z_{i-1})\phi(z_i \cdots z_j)^\omega = \phi(z_1 \dots z_{i-1}).$$

If we now set

$$e = \phi(z_i \cdots z_j)^\omega$$

$$s_{2n-1} = e \cdot \phi(z_{j+1} \cdots z_{|M|}), \text{ and } s_i = \phi(w_i) \text{ for } i < 2n-1,$$

we obtain, from the identity $e \cdot M_e \cdot e = e$,

$$\begin{aligned}
\phi(z') &= \phi(z_1 \dots z_{i-1}) \cdot \phi(z_{j+1} \cdots z_{|M|}) \\
&= \phi(z_1 \dots z_{i-1}) e \cdot e \phi(z_{j+1} \cdots z_{|M|}) \\
&= \phi(z_1 \dots z_{i-1}) e \cdot \big(\phi(z_{j+1} \cdots z_{|M|})(s_1 \cdots s_{2n-1})^{\omega-1} s_1 \cdots s_{2n-2}\big) \cdot e \phi(z_{j+1} \cdots z_{|M|}) \\
&= \phi(z_1 \dots z_{i-1}) e \cdot \phi(z_{j+1} \cdots z_{|M|})(s_1 \cdots s_{2n-1})^\omega \\
&= \phi(z') \cdot (s_1 \cdots s_{2n-1})^\omega.
\end{aligned}$$

The third equality above holds because by Lemma 7, $z_i \cdots z_j$ contains all the letters that occur in the z_k and the w_k, and hence all the values we inserted between occurrences of e belong to M_e.

Similarly, using the part of Lemma 7 concerning the prefix of y, we find a value s_{2n} such that $\phi(y') = (s_{2n} s_1 \cdots s_{2n-2})^\omega \phi(y')$. Since $M \models (u_n = v_n)$ we obtain

$$\begin{aligned}
\phi(zW_{n-1}y) &= \phi(z'')\phi(z')\phi(W_{n-1})\phi(y')\phi(y'') \\
&= \phi(z'')\phi(z')(s_1 \cdots s_{2n-1})^\omega \phi(W_{n-1})(s_{2n} s_1 \cdots s_{2n-2})^\omega \phi(y')\phi(y'') \\
&= \phi(z'')\phi(z')(s_1 \cdots s_{2n-1})^\omega \phi(W'_{n-1})(s_{2n} s_1 \cdots s_{2n-2})^\omega \phi(y')\phi(y'') \\
&= \phi(z'')\phi(z')\phi(W_{n-1})\phi(y')\phi(y'') \\
&= \phi(zW'_{n-1}y).
\end{aligned}$$

But this means that $M_{P,S} \models (u_{n-1} = v_{n-1})$, as we required. ◀

We now turn to the proof of Lemma 7.

Proof of Lemma 7. By symmetry, we only need to treat the part concerning the suffix of z. Whenever we need to emphasize the dependence of ψ on the chosen subword length m, we will write it as ψ_m. Recall that $\psi(z) = \psi_K(z)$ is the set of subwords of length no more than K in z, and that $\psi(zb) = \psi(z)$ for all $b \in B$.

We will show that if $B \subseteq \alpha(z)$ and $\psi_T(zb) = \psi_T(z)$ for all $b \in B$, where

$$T = |M| \cdot (k^2 + k)/2,$$

and $k = |\alpha(z)|$, then z contains a suffix with the required properties. This gives the lemma, because $\psi_K(zb) = \psi_K(z)$ implies $\psi_T(zb) = \psi_T(z)$ for any $\alpha(z) \subseteq A$.

The proof is by induction on $|\alpha(z)|$. The base case occurs when $\alpha(z) = B$. Let $B = \{b_1, \ldots b_r\}$. By repeated application of $\psi_T(zb_i) = \psi_T(z)$ we find $(b_1 \cdots b_r)^{|M|}$, which has length $|M||B| \leq |M|(|B|^2 + |B|)/2$, is a subword of z. In this case we can simply take $z' = z$ and factor $z = z_1 \cdots z_{|M|}$, where each z_i contains one of the factors $b_1 \cdots b_r$ as a subword.

We thus suppose that $\alpha(z) = A'$ contains B as a proper subset. Let $N = |A'|$. We look at a subword of maximal length $t_1 \cdots t_p$ of z such that $\alpha(t_i) = A'$. We must have $p \geq 1$. If $p \geq |M|$, we can again take $z' = z$ and factor z as $z_1 \cdots z_{|M|}$, where each z_i contains t_i as a subword. If $p < |M|$, we let $s = |A'|$, then we write

$$t_1 \cdots t_p = a_1 a_2 \cdots a_{ps}, \text{ and } z = z_0 a_1 z_1 \cdots a_{ps} z_{ps}.$$

We further suppose that this factorization represents the leftmost occurrence of $a_1 \cdots a_{ps}$ as a subword of z, in other words that z_{ps} has maximum possible length for this property. Note that $\alpha(z_{ps})$ is a strict subset of A', for otherwise z would have contained a longer subword $t_1 \cdots t_{p+1}$ with $\alpha(t_i) = A'$. Thus $|\alpha(z_{ps})| \leq N - 1$. Set $T = |M| \cdot ((N-1)^2 + (N-1))/2$. We must have $\psi_T(z_{ps}b) = \psi_T(z_{ps})$ for all $b \in B$. If not, there is a subword u of z_{ps} of length less than T such that ub is not a subword of z_{ps}. However $t_1 \cdots t_p ub$ has length no more than

$$(|M| - 1) \cdot N + T < |M| \cdot (N + ((N-1)^2 + (N-1))/2) = |M| \cdot (N^2 + N)/2,$$

and is accordingly a subword of z, and thus there is a strictly earlier occurrence of $t_1 \cdots t_p$ as a subword of z, a contradiction. We can thus apply the inductive hypothesis to z_{ps} and conclude that z_{ps} contains a suffix of the required type. ◀

5 Collapse of the hierarchy

In the original model-theoretic study of the alternation hierarchy in $\mathrm{FO}^2[<]$, Weis and Immerman [21] and also Kufleitner and Weil [9] showed that while the hierarchy is strict, it collapses for each fixed-size alphabet. An algebraic proof of strictness was given in [17], using the identities that form the subject of the present paper. We can use similar techniques to prove the collapse result.

▶ **Theorem 8.** *Let $n > 0$. If $M \in \mathbf{DA}$ is generated by n elements, then $M \in \mathbf{V}_n$.*

The proof, which we omit, uses our main result Theorem 6 to conclude that if $M \in \mathbf{DA}$, then $M \models (u_N = v_N)$ for some N. We then use identity manipulation in $\mathbf{DA}$ to show that this implies $M \models (u_n = v_n)$, where n is the number of generators.

In particular for any fixed alphabet the quantifier alternation hierarchy collapses.

▶ **Corollary 9.** *Any language over a k-letter alphabet definable by a two-variable sentence is definable by one in which the number of quantifier blocks is k.*

6 General decidability results

Here we show that for arbitrary pseudovarieties $\mathbf{V}$, the operation $\mathbf{V} \mapsto \mathbf{V} ** \mathbf{J}$ preserves decidability. This of course implies our result (a consequence of Theorem 6) that the varieties $\mathbf{V}_j$ are all decidable, but Theorem 6 is a sharper result, since it gives explicit identities. As we remarked in the introduction, the general decidability result was originally proved by Steinberg [16], but not previously published. Our approach has the advantages both of being relatively elementary, and yielding explicit bounds on the complexity of membership testing.

We suppose that $\phi : A^* \to M$ is a surjective homomorphism onto a finite monoid. Let $N > 0$, we denote by $\ker_N \phi$ the category $\ker(\psi_N \circ \phi^{-1})$, where ψ_N is the natural projection of A^* onto the quotient $A^*/\sim_N$. We set

$$K = |M| \cdot (|A|^2 + |A|)/2$$

as in the statement of Lemma 7. With these notations we have:

▶ **Theorem 10.** *Let* $\mathbf{V}$ *be a pseudovariety of monoids.* $M \in \mathbf{V} ** \mathbf{J}$ *if and only if every base monoid of* $\ker_K \phi$ *is in* $\mathbf{V}$.

We omit the proof. The idea is this: By Proposition 4, if $M \in \mathbf{V} ** \mathbf{J}$, then there is some K' such that all the base monoids of $\ker_{K'} \phi$ are in $\mathbf{V}$. We use Lemma 7 to show that if $K' > K$, then every base monoid of $\ker_K \phi$ divides a direct product of base monoids in $\ker_{K'} \phi$, so the result follows from Theorem 5.

We can effectively compute all the objects and arrow classes of $\ker_K \phi$ from ϕ, and we can also take $A = M$ and ϕ to be the extension of the identity map on M to A^*. The theorem thus immediately implies:

▶ **Corollary 11.** *If* $\mathbf{V}$ *is a decidable pseudovariety of finite monoids, then so is* $\mathbf{V} ** \mathbf{J}$.

7 Conclusion

We have shown that the identities given in [17] indeed characterize $\mathbf{V}_n$. There is, of course, a one-sided semidirect product, which has been much more thoroughly studied. Our results, and their proofs, can all be adapted to one-sided products, with little modification. In this case, the hierarchy collapses at the second level: $\mathbf{J} * \mathbf{J} * \mathbf{J} = \mathbf{J} * \mathbf{J}$. (This fact is not new. It has long been known that the closure of $\mathbf{J}$ under one-sided products is the pseudovariety $\mathbf{R}$ of $\mathcal{R}$-trivial monoids, and Brzozowski and Fich [3] showed $\mathbf{R} = \mathbf{J}_1 * \mathbf{J}$.)

In their paper, Kufleitner and Weil [9] give a completely different characterization of the levels of $\mathrm{FO}^2[<]$. It would be nice to see a direct connection between these two approaches.

Acknowledgements We are grateful to Manfred Kufleitner, Benjamin Steinberg, and Pascal Weil for detailed discussions of this work, and to the anonymous referees for their helpful suggestions.

References

1 Jorge Almeida. *Finite Semigroups and Universal Algebra.* Series in Algebra. World Scientific, 1994.

2 Jorge Almeida and Pascal Weil. Profinite categories and semidirect products. *Journal of Pure and Applied Algebra*, 123(1-3):1–50, 1998.

3 Janusz A. Brzozowski and Faith E. Fich. Languages of R-trivial monoids. *J. Comput. Syst. Sci.*, 20(1):32–49, 1980.

4 Samuel Eilenberg. *Automata, Languages, and Machines Vol. 2*. Pure and applied mathematics. Academic Press, 1976.

5 Neil Immerman and Dexter Kozen. Definability with bounded number of bound variables. *Inf. Comput.*, 83(2):121–139, 1989.

6 Johan A. W. Kamp. *Tense logic and the theory of linear order*. PhD thesis, University of California, Los Angeles, 1968.

7 Kenneth Krohn, Richard Mateosian, and John Rhodes. Methods of the algebraic theory of machines. I: Decomposition theorem for generalized machines; properties preserved under series and parallel compositions of machines. *J. Comput. Syst. Sci.*, 1(1):55–85, 1967.

8 Manfred Kufleitner and Pascal Weil. On FO^2 quantifier alternation over words. In *Mathematical Foundations of Computer Science*, pages 513–524, 2009.

9 Manfred Kufleitner and Pascal Weil. The FO^2 alternation hierarchy is decidable. In *Computer Science Logic*, pages 426–439, 2012.

10 R. McNaughton and S. Papert. *Counter-free automata*. M.I.T. Press research monographs. M.I.T. Press, 1971.

11 Jean-Éric Pin. *Varieties of formal languages*. North Oxford Academic, 1986.

12 John L. Rhodes and Bret Tilson. The kernel of monoid morphisms. *Journal of Pure and Applied Algebra*, 62:227–268, 1989.

13 Marcel Paul Schützenberger. A remark on finite transducers. *Information and Control*, 4(2-3):185–196, 1961.

14 Marcel Paul Schützenberger. Sur le produit de concatenation non ambigu. *Semigroup Forum*, 13:47–75, 1976. 10.1007/BF02194921.

15 Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, 1975.

16 Benjamin Steinberg. *Decidability and Hyperdecidability of Joins of Pseudovarieties*. PhD thesis, University of California at Berkeley, 1998.

17 Howard Straubing. Algebraic characterization of the alternation hierarchy in $\mathrm{FO}^2[<]$ on finite words. In *Computer Science Logic*, pages 525–537, 2011.

18 Pascal Tesson and Denis Therien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages, Coimbra (Portugal) 2001*, pages 475–500. World Scientific, 2002.

19 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC*, pages 234–240, 1998.

20 Bret Tilson. Categories as algebra: An essential ingredient in the theory of monoids. *Journal of Pure and Applied Algebra*, 48(1-2):83–198, 1987.

21 Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Logical Methods in Computer Science*, 5(3), 2009.

Decidable classes of documents for XPath*

Vince Bárány[1,2], Mikołaj Bojańczyk[2], Diego Figueira[2,3], and Paweł Parys[2]

1 **TU Darmstadt, Darmstadt, Germany**
2 **University of Warsaw, Warsaw, Poland**
3 **University of Edinburgh, Edinburgh, United Kingdom**

Abstract

We study the satisfiability problem for XPath over XML documents of bounded depth. We define two parameters, called *match width* and *braid width*, that assign a number to any class of documents. We show that for all k, satisfiability for XPath restricted to bounded depth documents with match width at most k is decidable; and that XPath is undecidable on any class of documents with unbounded braid width. We conjecture that these two parameters are equivalent, in the sense that a class of documents has bounded match width iff it has bounded braid width.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic, F.4.3 Formal Languages, H.2.3 Languages

Keywords and phrases XPath, XML, class automata, data trees, data words, satisfiability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.99

1 Introduction

This paper is about satisfiability of XPath over XML documents, modelled as data trees. A data tree is a tree where every position carries a *label* from a finite set, and a *data value* from an infinite set. The data values can only be tested for equality.

XPath satisfiability. XPath can be seen as a logic for expressing properties of data trees. Here are some examples of properties of data trees that can be expressed in XPath: "every two positions carry a different data value", "if x and y are positions that carry the same data value, then on the path from x to y there is at most one position that has label b". Our interest in XPath stems from the fact that it is arguably the most widely used XML query language. It is implemented in XSLT and XQuery and it is used in many specification and update languages. Query containment and query equivalence are important static analysis problems, which are useful to query optimization tasks. These problems reduce to checking for *satisfiability*: Is there a document on which a given XPath query has a non-empty result? By answering this question we can decide at compile time whether the query contains a contradiction and thus the computation of the query (or subquery) on the document can be avoided. Or, by answering the query equivalence problem, one can test if a query can be safely replaced by another one which is more optimized in some sense (e.g., in the use of some resource). Moreover, the satisfiability problem is crucial for applications on security, type checking transformations, and consistency of XML specifications.

Our point of departure is that XPath satisfiability is an undecidable problem [9]. There are two main approaches of working around this undecidability.

* Partially supported by FET-Open Project FoX, grant agreement 233599.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 99–111

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

1. **Restrict the formulas.** The first way is to consider fragments of XPath that have decidable satisfiability. For example, fragments without negation or without recursive axes [1]; or fragments whose only navigation can be done downwards [7] or downwards and rightwards [6] or downwards and upwards [8]. However, even though all these restrictions yield decidable fragments, the most expressive ones have huge, non-primitive-recursive, complexity bounds.
2. **Restrict the models.** When proving undecidability of XPath satisfiability, for each Minsky machine one constructs an XPath formula φ, such that models of φ describe computations of the Minsky machine. XML documents that describe computations of Minsky machines seem unlikely to appear in the real world; and therefore it sounds reasonable to place some restrictions on data trees, restrictions that are satisfied by normal XML documents, but violated by descriptions of Minsky machines.

This paper is devoted to the second approach.

Comparison with tree width

The archetype for our research is the connection between tree width and satisfiability of guarded second order logic, over graphs. Guarded second-order logic is a logic for expressing properties of undirected graphs. A formula of guarded second-order logic uses a predicate $E(x, y)$ for the edge relation, and can quantify over nodes of the graph, sets of nodes of the graph, and subsets of the edges in the graph. Satisfiability of guarded second-order logic over graphs is undecidable (already first-order logic has undecidable satisfiability). However, the picture changes when one bounds the tree width of graphs.

- Bounded tree width is a sufficient condition for decidability. More precisely, for every $k \in \mathbb{N}$, one can decide if a formula of guarded second-order logic is satisfied in a graph of tree width at most k [5].
- Bounded tree width is also a necessary condition for decidability: if a set of graphs X has unbounded tree width, then it is undecidable if a given formula of guarded second-order logic has a model in X [12].

Our contribution

Our goal in this paper is to find a parameter, which is to XPath over data trees, what tree width is to guarded second-order logic over graphs. As candidates, we define two parameters, called the *match width* and the *braid width* of data trees. Our results are:

- Bounded match width is a sufficient condition for decidability. For every $k \in \mathbb{N}$, one can decide if a formula of XPath is satisfied in a data tree of match width at most k.
- Bounded braid width is a necessary condition for decidability: if a set of data trees X has unbounded braid width, then it is undecidable if a given formula of XPath has a model in X.

Observe that the two statements talk about two different parameters. We conjecture that bounded match width is equivalent to bounded braid width, but we are unable to prove this.

Bounded depth data trees. Our results are restricted to data trees with bounded depth, which we believe is relevant since the depth of XML documents is very small in practice. However, we believe that our results can be extended to arbitrary data trees.

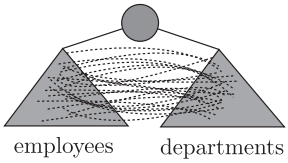
Why not tree width?

Instead of defining a new parameter for data trees, such as match width, why not simply work with tree width, which has proved so useful in other contexts? It is not difficult to

apply the notion of tree width to a data tree; for instance one can define the tree width of a data tree to be the tree width of the graph where the nodes are positions, and the edges are either tree edges (connecting a node to its parent in the tree), or data edges (connecting positions carrying the same data value). In fact, even guarded second-order logic (which is much more expressive than XPath) is decidable over data trees of bounded tree width.

So why not use tree width? A short answer is that because guarded second-order logic is much more powerful than XPath, one can define data trees which are difficult for guarded second-order logic (have high tree width), but easy for XPath (have low match width).

Another, related, point is that match width is a parameter that is more suited to analyzing data trees, with an emphasis on XML documents that store databases. A data tree contains two kinds of structure: the underlying tree structure, and the structure induced by comparing data values. When talking about the tree width of a data tree, these two kinds of structure are not distinguished, and merged into a single graph. Match width, on the other hand, is sensitive to the difference between the tree structure and the data structure. Consider the following example. Suppose that we have a data tree, which contains two subtrees: one subtree which contains employees, and one subtree which contains departments. Suppose that in each employee record, we have a pointer to a department record, which maps an employee to his/her department. Such a document is illustrated below, with the dashed lines representing links from employees to their departments.



As long as the sources of all the links (in the "employees" subtree) are located in a different subtree than the the targets of the links (in the "department" subtree), then the match width of the document will be small (as we shall prove in the paper), regardless of the distribution of the links. On the other hand, by appropriately choosing the distribution of the links (while still keeping all sources in the "employees" subtree and all targets in the "departments" subtree), the tree width of a document can be made arbitrarily high. In other words, the documents like the one in the picture are a class of data trees, where the match width is bounded, and the tree width is unbounded. (We also believe that this class of documents is rather typical for XML documents occurring in practice.) In particular, on this class of documents, guarded second-order logic is undecidable, while XPath is decidable.

Plan of the paper

In the next section we state the main definitions: data trees, data words, and class automata. Our main contributions are stated in terms of data words with at most two elements in each equivalence class, called *match words*, since it simplifies proofs and definitions. However, in later sections we show how to generalize our results to bounded depth data trees with unbounded number of elements per data equivalence class. We introduce *match width* in Section 3, and in Section 4 we show our main result, that class automata (or XPath) over classes of match words with bounded match width is decidable. In Section 5 we introduce *braid width* and show that XPath and class automata are undecidable over any class with unbounded braid width. In Sections 6 and 7 we extend the definitions and results to data words and to bounded depth data trees.

2 Preliminaries

Data trees and data words

We work with unranked trees, where the siblings are ordered. A tree over alphabet A is a tree where the nodes are labelled by letters from A. A *data tree* can be defined in two ways. The first way is that it is an unranked tree, where every node carries a label from the input alphabet A, and a data value from a infinite set (say, the natural numbers). The logics that we use can only compare the data values for equality, and therefore the only thing that needs to be known about data values in a data tree is which ones are equal. That is why we choose to model a data tree as a pair $(t, \sim)$, where t is a tree over the alphabet A, and $\sim$ is an equivalence relation on the nodes of t, which says which nodes have the same data value. Similarly, a *data word* is a pair $(w, \sim)$, where w is a word over the alphabet A and $\sim$ is an equivalence relation on the positions of w.

XPath

By XPath we mean the fragment capturing the navigational aspects of XPath 1.0. (called FOXPath in [2]). Expressions of this logic can navigate the tree by composing binary relations from a set of basic relations (a.k.a. *axes*): the parent relation (here noted $\uparrow$), child ($\downarrow$), ancestor ($\uparrow^*$), descendant ($\downarrow_*$), next sibling to the right ($\rightarrow$) or to the left ($\leftarrow$), and their transitive closures ($\rightarrow^*$, ${}^*\leftarrow$). For example, $\alpha = \uparrow[a]\uparrow\downarrow[b]$, defines the relation α between two nodes x, y such that y is an uncle of x labeled b and the parent of x is labeled a. Boolean tests are built by using these relations. An expression like $\langle\alpha\rangle$ (for a relation α) tests that there exists a node accessible with the relation α from the current node. Most importantly, a data test like $\langle\alpha = \beta\rangle$ (resp. $\langle\alpha \neq \beta\rangle$) tests that there are two nodes reachable from the current node with the relations α and β that have the same (resp. different) data value. A formal definition of the considered fragment of XPath can be found in [2]. XPath can be also interpreted over data words, using the sibling axes.

Class automata

In the technical parts of the paper, we will use an automaton model for XPath, which is called *class automata*, introduced in [4]. Suppose that t is a tree over a finite alphabet A, and X is a set of nodes in t. We write $t \otimes X$ for the tree over alphabet $A \times 2$ (this is the same as $A \times \{0, 1\}$) obtained from t by extending the label of each node by a bit, which indicates whether the node belongs to X. Similarly, for any two trees t_1, t_2 with the same nodes and labelled by alphabets A_1 and A_2 respectively, we write $t_1 \otimes t_2$ for the tree over alphabet $A_1 \times A_2$, which has the same nodes as t_1 and t_2, and where each node is labelled by the pair of labels from t_1 and t_2.

A *class automaton* consists of

- an input alphabet A;
- a work alphabet B;
- a class condition L, which is a regular language of trees over the alphabet $A \times B \times 2$.

The automaton accepts a data tree $(t, \sim)$ if there is some tree s over the work alphabet B with the same nodes as t, such that $t \otimes s \otimes X \in L$ for every equivalence class X of $\sim$.

It was shown in [4] that class automata capture XPath.

▶ **Theorem 1** ([4]). *For every XPath boolean query, one can compute a class automaton, which accepts the same data trees.*

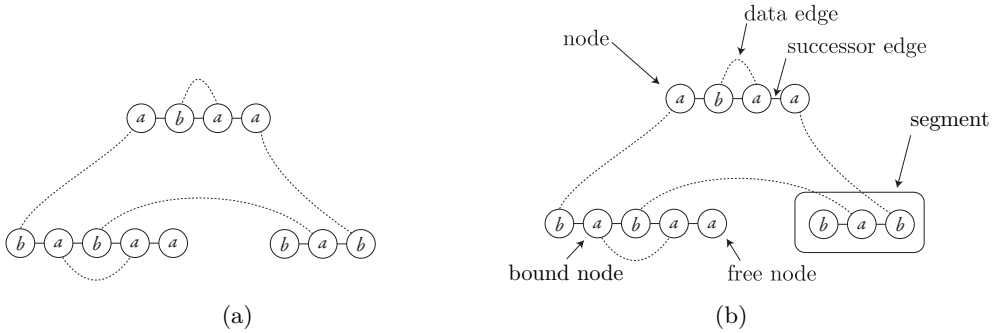


Figure 1 A split match word (a) and its parts (b).

3 Match width

In this section we define the match width of a data word. In fact, we will work with data words whose every class has size at most 2, that we call *match words*. Later on, we will comment on how to extend this definition to data words and data trees.

A *split match word* is a finite set of words, together with an additional set of edges, which connect positions of these words with each other. Figure 1 (a) contains an example of a split match word. We write τ for split match words. The words are called the *segments* of the split word.

We can think of a split word as a graph where the nodes are positions of the words, with two types of edges: *successor edges*, which go from one position to the next in each segment, and *data edges*, which are the additional edges. We require the data edges to be a matching. The matching need not be perfect: some nodes might not have an outgoing data edge, such nodes are called *free nodes*. The nodes that are not free are called *bound nodes*. These definitions are illustrated in Figure 1 (b). Note that a match word can be seen as a special case of a split match word, which has one segment.

3.1 Operations on split match words

We will construct split match words out of simpler split match words, using the following four operations.

- **Base** (no arguments). We can begin with a split match word which has one segment, with one position, and no data edges.
- **Union** (two arguments). We can take a disjoint union of two split match words. In the resulting split word, the number of segments is the sum of the numbers of segments of the arguments; and there are no data edges between segments from different arguments.
- **Join** (one argument). We can add a successor edge, joining two segments into one segment.
- **Match** (one argument). We can choose two distinct segments, and add any set of data edges with sources in one segment and targets in the other segment.

The operations (except union) are not deterministic (in the sense that the result is not uniquely determined by the operation name and the argument). The result of base depends on the choice of a label on the only position; the result of applying join to a split match word also depends on the choice of segments to be joined; and the result of applying match depends on the choice of new data edges.

Derivation. A *derivation* is a finite tree, where each node is of one of four types (base, union, join or match) and is labelled by a split match word, satisfying the following natural

property. The leaves are of base type, nodes of union type have two children, and nodes of join and match type have one child. Suppose that x is a node in a derivation. Then the split match word in the label of node x is constructed, using the operation in the type of x, from the split match words in the children of x. A derivation is said to *generate* the split match word that labels its root.

Rank. Suppose that t is a derivation. To each node of t, we assign a number, which is called the *rank* of the node. The rank of a base node is 0. The rank of a union or join node is the maximum of the ranks of the children. The rank of a match node is 0 if the split match word in the label of x has no free nodes; otherwise it is $n+1$, where n is the rank of the unique child of x.

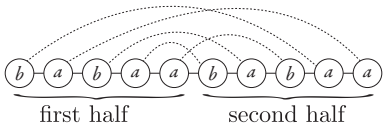
Width. The *rank* of a derivation is the maximal rank that appears in a node of the derivation. The *segment size* of a derivation is the maximal number of segments that appear in a split match word in one of the nodes of the derivation. Finally, the *width* of a derivation is the maximum of its rank and segment size. The *match width* of a split match word is the minimal width of a derivation that generates it. We are most interested in the special case of the match width of a match word, seen as a special case of a split match word.

In order to have small width, a derivation needs to have both small rank and small segment size. In the following examples we show that requiring only the rank to be small, or only the segment size to be small, is not restrictive enough, since all match words can be generated that way.

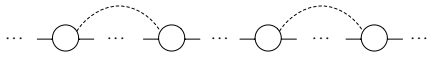
▶ **Example 2.** Every split match word with one segment (which is the same thing as a match word, with possibly some free nodes that are not matched) can be generated by a derivation of segment size 2, but possibly unbounded rank. The proof is by induction on the number of nodes. Consider then a split match word with $n+1$ nodes. Apply the induction assumption to the first n nodes. Now add the $(n+1)$-st node as a separate segment using a union rule, connect it if necessary with one of the first n nodes using a match rule, and then join it using the join rule.

▶ **Example 3.** Every match word can be generated by a derivation of rank 1, but possibly unbounded segment size. We prove the following claim: every split match word τ without free nodes can be generated by a derivation of rank 1, but possibly unbounded segment size. The proof is by induction on the number of nodes in τ. For the induction step, consider a split match word τ. Choose some data edge in τ, which connects nodes x and y. Create a new split match word, call it σ, by removing the nodes x and y together with their incident successor edges and the data edge connecting x and y. (Observe that σ can have more segments than τ; for example if x and y are in separate segments, and they are not the first or last positions in those segments, then σ will have two more segments than τ, since the segments containing x and y will break into two segments each.) The new split match word σ has no free nodes; and therefore by induction assumption it has a derivation of rank 1. Since there are no free nodes, the root of the derivation has rank 0. Therefore, we can add x and y as separate segments, connect them with an edge, and then join them to σ creating τ. Using this claim, we can add free nodes and join all segments, creating any match word.

▶ **Example 4.** A segregated match word is a data word of even length $2n$, such that every data edge connects a node from the first half with a node in the second half.



(a) A segregated word.



(b) The forbidden pattern.

Clearly, segregated words have match width at most 2. They can be alternatively described as those data words not containing four elements in the configuration shown under (b).

In [4] it was shown that emptiness for class automata is decidable over segregated data words. It is instructive to recall the argument in anticipation of the broader technique underlying our main result. For the rest of this argument we say that a match word is *free* if no data value occurs twice in it, i.e. if it is entirely devoid of data equality edges and thus, as long as it is considered in isolation, it is hardly more than a word over a finite alphabet. It is easy to see that it is sufficient to just consider a fixed, one-letter work alphabet $B = \{b\}$. Let $\mathcal{A}$ be a class automaton with a class condition $L \subseteq (A \times B \times 2)^*$ recognized by a monoid morphism $\alpha : (A \times B \times 2)^* \to M$ so that $L = \alpha^{-1}(F)$ for some $F \subseteq M$. We define the *α-profile*, $\pi_\alpha(w)$, of a free match word w as the vector $X \in \mathbb{N}^M$ whose every entry X_m equals the number of positions x in w such that $\alpha(w \otimes b^{|w|} \otimes \{x\}) = m$. Observe that the set $\Lambda_\alpha = \{\pi_\alpha(w) \mid w \text{ free match word}\}$ of all profiles of free match words is the Parikh image of a suitable regular language, whence, by Parikh's theorem [11], it is semi-linear (Presburger definable [10]). Indeed, for each free match word w let $\sigma(w)$ be the word of the same length satisfying $\sigma(w)[i] = \alpha(w \otimes b^{|w|} \otimes \{i\})$ for each $1 \leq i \leq |w|$. Then $\pi_\alpha(w)$ is precisely the image of $\sigma(w)$ under the Parikh mapping. To see that the set $\{\sigma(w) \mid w \text{ a free match word}\}$ is regular it suffices to say that $\{w \times \sigma(w) \mid w \text{ a free match word}\}$ is recognizable by an automaton that aggregates in a straightforward manner the value of $\alpha(v \otimes b^{|v|} \otimes \{0\})$ for each prefix and each suffix v of w.

Whenever free match words u and v have the same length and share the same data values, then the segregated word $w = uv$ is accepted by the class automaton $\mathcal{A}$ if, and only if, there is a matrix $X \in \mathbb{N}^{M \times M}$ such that for all $r, s \in M$ $X_{r,s} \neq 0$ only if $r \cdot s \in F$ and $\sum_s X_{r,s} = Y_r$ and $\sum_r X_{r,s} = Z_s$, where $Y = \pi_\alpha(u)$ and $Z = \pi_\alpha(v)$. Here X represents a family of matching by prescribing how many positions of each type within u should be matched to how many positions of whichever type within v.

In conclusion, the class automaton α accepts some segregated word iff the above system of linear equations has a solution for $\{Y_r, Z_s, X_{r,s}\}_{r,s \in M}$. The set of solutions is definable in Presburger arithmetic (viz. semilinear [10]) and can be effectively verified for emptiness.

Main result

We now announce the main results of this paper: emptiness for class automata is decidable over match words of bounded match width.

▶ **Theorem 5.** *The following problem is decidable:*
Input *A number n and a class automaton.*
Question *Does the automaton accept some match word of match width $\leq n$?*

In Section 4, we sketch the proof of the theorem. Before proving the theorem, we further discuss the notion of match width, and show how it is related to tree width.

Match width and first-order logic. Match width is a measure that is adapted specifically for class automata. If we consider monadic second-order logic, or even first-order logic, over match words of bounded match width, then satisfiability is undecidable.

▶ **Lemma 6.** *Satisfiability of first-order logic is undecidable over segregated match words, as defined in Example 4, which have match width at most 2.*

Match width and tree width. On the other hand, bounded tree width implies bounded match width, but the converse does not hold.

▶ **Lemma 7.** *If a match word has tree width k, then it has match width at most $3k + 3$.*

▶ **Lemma 8.** *Match words of match width 2 can have unbounded tree width.*

4 Bounded match width sufficient for decidability

In this section, we present a proof sketch of Theorem 5. In fact, we show a stronger result, Theorem 9, which implies Theorem 5.

Numbered segments. Consider a split match word τ with n segments. In this section, it will be convenient to number the segments, so we assume that each split match word comes with an implicit ordering of the segments. The segments will be written as $\tau[1], \ldots, \tau[n]$. We write $\text{segments}(\tau)$ for the set $\{1, \ldots, n\}$.

Type of a split match word.

Consider a monoid morphism $\alpha : (A \times 2)^* \to M$. We define the *$\alpha$-type of a split match word* τ to be the following information.

- The *empty type*, which is the vector $\text{emptytype}_\tau \in M^{\text{segments}(\tau)}$ that maps $i \in \text{segments}(\tau)$ to the type $\alpha(\tau[i] \otimes \emptyset) \in M$.
- The *free type*, which is the function $\text{freetype}_\tau : \text{segments}(\tau) \times M \to \mathbb{N}$ that maps (i, m) to the number of free positions x in the word $\tau[i]$ that satisfy $\alpha(\tau[i] \otimes \{x\}) = m$.
- The *bound type*, which is the set $\text{boundtype}_\tau \subseteq M^{\text{segments}(\tau)}$ which contains a vector $v \in M^{\text{segments}(\tau)}$ if there is some matched pair of positions, call them x, y, such that on coordinate $i \in \text{segments}(\tau)$, the vector has the value $\alpha(\tau[i] \otimes \{x, y\})$. Note that it may be that positions x, y are not in component i, or that only one is in the component.

When the morphism α is clear from the context, we say type instead of α-type. The type is therefore some finite information (the empty type and the bound type), together with a vector of natural numbers (the free type). Because of the free type, the set of possible α-types is potentially infinite. However, as we shall see below, semilinear sets can be used to represent the vectors in the free type, at least as long as the match width is bounded. We briefly recall semilinear sets below.

Semilinear sets

A *linear space* is $\mathbb{N}^k$ for some dimension $k \in \mathbb{N}$. A *semilinear space* is a finite disjoint union of linear spaces. The components of the disjoint union are called the *components* of the semilinear space. We write semilinear spaces as $\coprod_{i \in I} X_i$, where the index set I is finite, each component X_i is a linear space, and $\coprod$ stands for disjoint union. Semilinear spaces are closed under Cartesian products and disjoint unions.

A *semilinear subset* of a linear space is defined in the usual way. That is, as a finite union of linear subsets, where the linear subsets of $\mathbb{N}^k$ are those of the following form: $\mathbf{v_0} + \sum_{j=1}^{t} \mathbb{N}\mathbf{v_j} = \{\mathbf{v_0} + \sum_{j=1}^{t} n_j \mathbf{v_j} : n_1, \ldots, n_t \in \mathbb{N}\}$, for some t and fixed $\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_t} \in \mathbb{N}^k$. A *semilinear subset* of a semilinear space associates a semilinear subset to each of the components.

Representing α-types.

We are now ready to state the main technical result used in the proof of Theorem 5. Fix a number k of segments. An α-type of a split match word with k segments is an element of

$$X_k \stackrel{\text{def}}{=} \underbrace{M^k}_{\text{empty type}} \times \underbrace{\mathbb{N}^{k \times M}}_{\text{free type}} \times \underbrace{P(M^k)}_{\text{bound type}} .$$

The above is a semilinear space, with one linear space of dimension $k \times M$ for every pair of empty and bound types. Therefore, it makes sense to ask if the set of possible α-types is semilinear or not. The answer is positive, when the match width is bounded, as the following theorem shows.

▶ **Theorem 9.** *The set* $\{\text{type}_\tau : \tau \text{ is a } k\text{-segment split match word of match width} \leq n\}$, *where* $n \in \mathbb{N}$, *is a semilinear subset of* X_k *and can be computed.*

The proof of the theorem uses Parikh's theorem, which says that the Parikh image of a context-free language is semilinear. Here we only show that Theorem 9 implies Theorem 5.

Proof of Theorem 5. Recall that the theorem says that one can decide if a class automaton accepts some match word of match width $\leq k$. Let A be the input alphabet, B be the work alphabet, and $L \subseteq (A \times B \times 2)^*$ the class condition of the class automaton. Let $\alpha : (A \times B \times 2)^* \to M$ be a monoid morphism which recognizes the language L. In other words there is some set $F \subseteq M$ such that L is the inverse image $\alpha^{-1}(F)$.

The match width of a word does not change after its positions have been additionally labelled by the work alphabet B. Therefore, we want to decide the following question: is there some match word $(w, \sim)$ over the alphabet $A \times B$, with match width $\leq k$, and so that

- $w \otimes \{x, y\} \in L$ for every matched positions $\{x, y\}$ in $\sim$, and
- $w \otimes \{x\} \in L$ for every free position $\{x\}$ in $\sim$.

The above condition says that there is a split match word with one segment such that the free type maps every pair (i, m) with $m \notin F$ to 0 (i.e. all free positions x are so that $w \otimes \{x\} \in L$), and the bound type is a subset of F. By Theorem 9, we can compute all possible α-types of match words with one segment, and test if there is some such α-type. ◀

5 A necessary condition for decidability

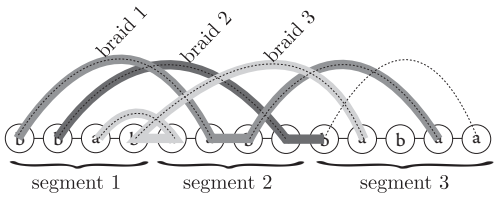
So far, we have defined a measure of match words, namely match width. Bounded match width is a sufficient condition for decidable emptiness of class automata. In this section, we define a complementary measure, called *braid width*, such that emptiness of class automata is undecidable on any class of match words with unbounded braid width. Therefore, having bounded braid width is a necessary condition for decidable emptiness of class automata.

Definition of braid width

Consider a match word, interpreted as a graph with successor edges and data edges. Split the match word into n consecutive subwords, henceforth called *segments*. Consider a path, which uses successor and data edges, and visits nodes $x_1, \ldots, x_k$. Such a path is a *braid* if:

- the path visits all n segments, and
- if a node of the path is in segment i, then subsequent nodes on the path cannot revisit any of the segments $1, \ldots, i-2$.

(The notion of a braid is relative to some decomposition into segments.) We say that a match word has *braid width* at least n if it can be split into n segments, such that one can find n node-disjoint braids. Below there is a picture of a match word with braid width at least 3, along with the witnessing segments and braids.



Notice that a braid can visit the previous segment. In fact, this is necessary, as there are classes of match word with unbounded braid width that, if we would restrict the braids to go only forward, would otherwise have bounded braid width. Also, for any fixed $k \in \mathbb{N}$ one can construct a class automaton $\mathcal{A}_k$ recognizing all match words of braid width at least k.

▶ **Theorem 10.** *Let X be a class of match words of unbounded braid width. Then, emptiness of class automata is undecidable on X. If in addition X is closed under arbitrary relabellings, then even XPath is undecidable on X.*

Proof idea. Observe that the first claim follows from the second, since class automata capture XPath by Theorem 1. The second claim is proved by reduction from the halting problem for 2-counter Minsky machines. To every Minsky machine $\mathcal{M}$ we associate a boolean XPath query whose models are match words that code accepting runs of $\mathcal{M}$. It is not hard to see that, for any k, there must be a word in X with k braids and k segments such that:

1. every braid begins in the first segment, and finishes in the last segment, and
2. consecutive nodes on every braid are either in the same or in neighboring segments.

Such a match word with an appropriate labelling will represent a run of $\mathcal{M}$ of length k so that $n_1 + n_2 \leq k$, where n_i is the maximum value of counter i reached during the run.

We use labels to mark which are the nodes involved in the k braids, and for every such node in a braid, where to find the previous and next node in the traversal of the braid (it could be: the node to the left or to the right, or the node in the same equivalence class). Also, labels mark the beginning and end of each segment; and for every segment and braid, we add a special label to flag the first node of the segment in the braid traversal.

An XPath formula can verify that the labels correctly code braids with the properties 1 and 2. Now, making use of the fact that there are exactly k flagged nodes in every segment, and that there is a way of linking these k nodes in a segment with the k flagged nodes of the next segment, we can code configurations of $\mathcal{M}$. In order to code the counters, each braid is associated to one of the counters through a label, and each flagged element is labelled with being active or inactive. The value of counter $i \in \{1, 2\}$ is then the number of active flagged nodes of braids associated to counter i inside the segment. If we further label each segment with a state, each segment codes a configuration of $\mathcal{M}$. Depending on the instruction, an XPath formula can test whether a counter is 0: no active flagged elements in the segment. Further, a formula can make a counter to increase (or decrease) its value: we choose an inactive flagged node in the segment and we force that the next flagged node in the braid traversal is active, while preserving the active/inactive state of all remaining flagged nodes.

Each of the above properties is expressible in XPath and we can therefore ensure that an accepting run of $\mathcal{M}$ is encoded in the match word. ◀

Figure 2 An example of a data expansion $(w, \sim)$ of a data word $(w', \sim')$.

6 Data words instead of match words

We discuss how the definition of braid and match width can be generalized to classes of data words with arbitrarily many elements in each equivalence class. The same decidability and undecidability results can also be transfered to this general definition.

Suppose a data word $(w, \sim)$ with two data classes X, Y with at least two elements each, so that the rightmost position of X is x and the leftmost position of Y is $x+1$ (i.e., the next position of x). We say that $(w, \sim)$ is a *data expansion* of $(w', \sim')$, if $(w', \sim')$ is the result of removing $x + 1$ from $(w, \sim)$ and joining the data classes X and Y (cf. Figure 2). We say that a match word $(w, \sim)$ is a *match expansion* of $(w', \sim')$ if it is the result from applying repeatedly data expansions to $(w', \sim')$. The class of match words C is the match expansion of a class of data words C', if C consists of all match expansions of data words from C'.

We define that a class C of data words has match width at least/at most n if the match expansion of C has match width at least/at most n. Similarly, C has braid width at least/at most n if the match expansion of C has braid width at least/at most n.

These definitions allow us to transfer our decidability and undecidability results for class automata. Further, the notions of bounded match width and bounded braid width coincide on match words if and only if they coincide on data words.

▶ **Lemma 11.** *For any class of data words with unbounded braid width, the emptiness problem for class automata is undecidable.*

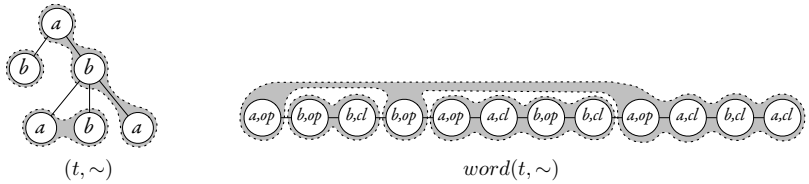
▶ **Lemma 12.** *Given a class automaton $\mathcal{A}$ and a number n, the emptiness problem for $\mathcal{A}$ restricted to data words with match width at most n is decidable.*

Note that, by Theorem 1, Lemma 12 is also true for XPath. However, we do not know if the undecidability result for XPath (or even *regular*-XPath) of Theorem 10 holds for classes of data words of unbounded braid width.

7 Data trees instead of data words

In the previous sections we discussed the case of data words. But what about data trees? Here we extend the notions of braid width and match width to data trees and generalize our decidability and undecidability results.

One solution is to reduce data trees to words. For a data tree $(t, \sim)$, we define its word representation $word(t, \sim)$, which is like the text representation of an XML tree. If the labels of t are A, then the labels of $word(t, \sim)$ are $\{open, close\} \times A$. Every node of $(t, \sim)$ corresponds to two nodes in $word(t, \sim)$, one with an opening tag and one with a closing tag, with the same data value, as depicted below.



If the depth of the tree is known in advance, and can be encoded in the states of an automaton, then this representation can be decoded by a class automaton.

▶ **Lemma 13.** *Fix $d \in \mathbb{N}$. For any class automaton on data trees $\mathcal{A}$, one can compute a class automaton on data words $\mathcal{A}_d$ such that $\mathcal{A}$ accepts a data tree $(t, \sim)$ if and only if $\mathcal{A}_d$ accepts the data word $word(t, \sim)$ provided that $(t, \sim)$ has depth at most d.*

We extend the definition of match and braid width to data trees following the $(t, \sim) \mapsto word(t, \sim)$ coding. A class of data trees has braid/match width of at least/at most n if its data word representation has braid/match width of at least/at most n. In view of the lemma above, we have the following.

▶ **Lemma 14.** *For any class automaton on data trees $\mathcal{A}$ and numbers d, n, the emptiness problem for $\mathcal{A}$ restricted to data trees of depth at most d and match width at most n is decidable.*

8 Discussion

We conjecture that match width and braid width are equivalent in the sense that a class of data words has bounded match width if, and only if, it has bounded braid width. One implication of the conjecture follows from Theorems 5 and 10. Namely, for every k, the class of documents of match width at most k has bounded braid width, since otherwise the class would have undecidable emptiness for class automata. In other words, unbounded braid width means unbounded match width. The content of the conjecture is therefore the other implication: is it the case that unbounded match width means unbounded braid width?

We also conjecture that XPath is undecidable on unbounded match width data words or data trees, but we are unable to prove it.

Finally, we believe that the results can also be extended to arbitrary data trees, by defining accordingly split data trees and using forest algebra [3] instead of monoids.

References

1 Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *Journal of the ACM (JACM)*, 55(2):1–79, 2008.

2 Michael Benedikt and Christoph Koch. XPath leashed. *ACM Comput. Surv.*, 41(1), 2008.

3 M. Bojańczyk and I. Walukiewicz. Forest algebras. In *Automata and Logic: History and Perspectives*, pages 107–132. Amsterdam University Press, 2007.

4 Mikolaj Bojanczyk and Slawomir Lasota. An extension of data automata that captures XPath. In *LICS*, pages 243–252. IEEE Computer Society, 2010.

5 Bruno Courcelle. Graph rewriting: A bibliographical guide. In *Term Rewriting*, volume 909 of *Lecture Notes in Computer Science*, page 74. Springer, 1993.

6 Diego Figueira. Alternating register automata on finite data words and trees. *Logical Methods in Computer Science (LMCS)*, 8(1:22), 2012.

7 Diego Figueira. Decidability of downward XPath. *ACM Transactions on Computational Logic (TOCL)*, 13(4), 2012. To appear.

8 Diego Figueira and Luc Segoufin. Bottom-up automata on data trees and vertical XPath. In *International Symposium on Theoretical Aspects of Computer Science (STACS'11)*. Springer, 2011.

9 Floris Geerts and Wenfei Fan. Satisfiability of XPath queries with sibling axes. In *International Symposium on Database Programming Languages (DBPL'05)*, volume 3774 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 2005.

10 Seymour Ginsburg and Edwin H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

11 Rohit J. Parikh. On context-free languages. *J. ACM*, 13:570–581, October 1966.

12 D. Seese. The structure of the models of decidable monadic theories of graphs. *Ann. Pure Appl. Logic*, 53(2), 1991.

Faster Deciding MSO Properties of Trees of Fixed Height, and Some Consequences

Jakub Gajarský and Petr Hliněný*

Faculty of Informatics, Masaryk University, Brno, Czech Republic
{xgajar,hlineny}@fi.muni.cz

Abstract

We prove, in the universe of trees of bounded height, that for any MSO formula with m variables there exists a set of kernels such that the size of each of these kernels can be bounded by an elementary function of m. This yields a faster MSO model checking algorithm for trees of bounded height than the one for general trees. From that we obtain, by means of interpretation, corresponding results for the classes of graphs of bounded tree-depth (MSO_2) and shrub-depth (MSO_1), and thus we give wide generalizations of Lampis' (ESA 2010) and Ganian's (IPEC 2011) results. In the second part of the paper we use this kernel structure to show that FO has the same expressive power as MSO_1 on the graph classes of bounded shrub-depth. This makes bounded shrub-depth a good candidate for characterization of the hereditary classes of graphs on which FO and MSO_1 coincide, a problem recently posed by Elberfeld, Grohe, and Tantau (LICS 2012).

1998 ACM Subject Classification G.2.2 Graph Theory, F.4.1 Mathematical Logic

Keywords and phrases MSO graph property; tree-width; tree-depth; shrub-depth

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.112

1 Introduction

First order (FO) and monadic second-order (MSO) logics play undoubtedly crucial role in computer science. Besides traditional tight relations to finite automata and regular languages, this is also witnessed by their frequent occurrence in the so called *algorithmic metatheorems* which have gained increasing popularity in the past few years. The term algorithmic metatheorem commonly refers to a general algorithmic toolbox ready to be applied onto a wide range of problems in specific situations, and MSO or FO logic is often used in the expression of this "range of problems".

One of the perhaps most celebrated algorithmic metatheorems (and the original motivation for our research) is Courcelle's theorem [3] stating that every graph property ϕ expressible in the MSO_2 *logic of graphs* (allowing for both vertex and edge set quantifiers) can be decided in linear FPT time on graphs of bounded tree-width. Courcelle, Makowsky, and Rotics [4] then have analogously addressed a wider class of graphs, namely those of bounded clique-width, at the expense of restricting ϕ to MSO_1 *logic* (i.e., with only vertex set quantification). Among other recent works on algorithmic metatheorems we just briefly mention two survey articles by Kreutzer [16] and by Grohe–Kreutzer [15], and an interesting recent advance by Dvořák, Král', and Thomas [7] showing linear-time FPT decidability of FO model checking on the graphs of "bounded expansion".

Returning back to Courcelle's theorem [3] and closely related [1, 4], it is worth to remark that a solution can be obtained via interpretation of the respective graph problem into an

* The authors have been supported by the Czech Science Foundation, project no. P202/11/0196.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 112–123

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

MSO formula over coloured trees (which relates the topic all the way back to Rabin's S2S theorem [21]). However, a drawback of these metatheorems is that, when their runtime is expressed as $\mathcal{O}\big(f(\phi, width(G)) \cdot |G|\big)$, this function f grows asymptotically as $2^{\left.2^{\cdot^{width(G)}}\right\}a}$ where the height a depends on ϕ, precisely on the quantifier alternation depth of ϕ (i.e., f is a *non-elementary function* of the parameter ϕ). The latter is not surprising since Frick and Grohe [11, 10] proved that it is not possible to avoid a non-elementary tower of exponents even in deciding MSO properties on all trees or coloured paths (unless P=NP).

Given the importance of Courcelle's and other related algorithmic metatheorems, it is a bit of surprise that apparently no research paper tackled this "nonelementary exponential tower" issue of deciding graph MSO properties until recently: The first step in this direction occurred in a 2010 ESA paper by Lampis [17], giving an FPT algorithm for MSO_2 model checking on graphs of bounded vertex cover with only a double-exponential parameter dependence. Ganian [13] then analogously addressed MSO_1 model checking problem on graphs of bounded so-called twin-cover (much restricting bounded clique-width).

MSO on trees of bounded height

Frick–Grohe's negative result leaves main room for possible improvement on suitably restricted subclass(es) of all coloured trees, namely on those avoiding long paths. In this respect, our first result here (Theorem 3.2 and Corollary 3.3) gives a new algorithm for deciding MSO properties ϕ of rooted m-coloured trees T of fixed height d. This algorithm uses so called kernelization—which means it efficiently reduces the input tree into an equivalent one of elementarily bounded size, leading to an FPT algorithm with runtime

$$\mathcal{O}\left(|V(T)|\right) + 2^{\left.2^{\cdot^{\mathcal{O}(m|\phi|^2)}}\right\}d+1}.$$

Informally, our algorithm "trades" quantifier alternation of ϕ for bounded height of the tree. Hence there is nothing interesting brought for all trees, while on the other hand, our algorithm presents an improvement over the previous on the trees of height $\leq d$ for every fixed value d. We refer to Section 3.1 for details and exact expression of runtime.

In a more general perspective, our algorithm can be straightforwardly applied to any suitable "depth-structured" graph class via efficient interpretability of logic theories. This includes the aforementioned results of Lampis [17] and Ganian [13] as special cases. We moreover extend the algorithm (Theorem 3.4) to testing MSO_2 properties on all graphs of *tree-depth* $\leq d$ (see Definition 2.1) in elementary FPT, covering a much wider graph class than that of bounded vertex cover. This in Section 3.2 concludes the first half of our paper.

Expressive power of FO and MSO

Secondly, the existence of an (elementarily-sized) kernel for MSO properties ϕ of trees of fixed height d (Theorem 3.2) is interesting on its own. Particularly, it immediately implies that any such MSO sentence ϕ can be equivalently expressed in FO on the trees of height d (simply testing the finitely many bounded-size kernels for which ϕ is true, Theorem 4.1). This brings us to the very recent paper of Elberfeld, Grohe, and Tantau [9] who proved that FO and MSO_2 have equal expressive power on the graphs of bounded tree-depth. Their approach is different and uses a constructive extension of Feferman–Vaught theorem for unbounded partitions. We can now similarly derive the result from Theorem 3.2, as in the tree case.

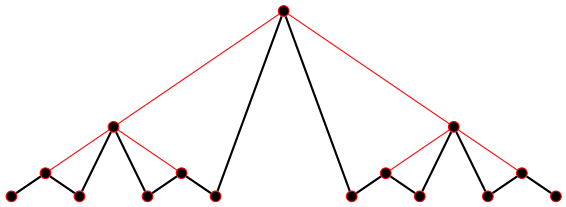


Figure 1 The path of length 14 has tree-depth $3 + 1 = 4$ since it is contained in the closure of the depicted (red) tree of height 3. It can be proved that this is optimal.

Going a step further, we actually half-answer the main open question posted in [9]; what characterizes the hereditary graph classes on which the expressive powers of FO and MSO_1 coincide? We use Theorem 3.2 and the new notions of [14] to prove that FO and MSO_1 coincide (Theorem 4.3) on all graph classes of bounded so called *shrub-depth* (see Definition 2.3). Unfortunately, due to lack of a suitable "forbidden substructure" characterization of shrub-depth, we are not yet able to prove the converse direction, but we conjecture that a hereditary class on which FO and MSO_1 coincide must have bounded shrub-depth (Conjecture 4.4). This conjecture is also supported by the following claim in [14]; a graph class $\mathcal{C}$ has an MSO_1 interpretation in the class of coloured trees of height $\leq d$ iff $\mathcal{C}$ is of shrub-depth $\leq d$.

2 Preliminaries

We assume standard terminology and notation of graph theory, see e.g. Diestel [5]. Due to limited space, we refer there [5] for the standard definition of tree-width $tw(G)$.

For an introduction to parameterized complexity we suggest [6]. Now we just recall that a problem $\mathcal{P}$ with an input $\langle x, k\rangle \in \Sigma^* \times \mathbb{N}$ is *fixed parameter tractable*, or FPT, if it admits an algorithm in time $\mathcal{O}\big(f(k) \cdot |x|^{\mathcal{O}(1)}\big)$ where f is an arbitrary computable function. It is known that $\mathcal{P}$ is in FPT if, and only if, it has a *kernel*, i.e., every instance $\langle x, k\rangle$ can be in polynomial time transformed to an equivalent instance $\langle x', k'\rangle$ such that $\langle x, k\rangle \in \mathcal{P} \iff \langle x', k'\rangle \in \mathcal{P}$ and $|\langle x', k'\rangle| \leq g(k)$ for some computable g.

Measuring depth of graphs

Our paper deals with some not-so-known decompositions of graphs, too. The first one is related to tree-decompositions of low depth.

▶ **Definition 2.1** (Tree-depth [18]). The *closure* $cl(F)$ of a rooted forest F is the graph obtained from F by adding from each node all edges to its descendants. The *tree-depth* $td(G)$ of a graph G is one more than the smallest height (distance from the root to all leaves) of a rooted forest F such that $G \subseteq cl(F)$.

Note that tree-depth is always an upper bound for tree-width. Some useful properties of it can be derived from the following asymptotic characterization: If L is the length of a longest path in a graph G, then $\lceil \log_2(L+2) \rceil \leq \mathrm{td}(G) \leq L + 1$. See Figure 1. For a simple proof of this, as well as for a more extensive study of tree-depth, we refer the reader to [19, Chapter 6]. Particularly, it follows that $td(G)$ can be approximated up to an exponential error by a depth-first search, and furthermore computed exactly in linear FPT using the tree-width algorithm of Bodlaender [2].

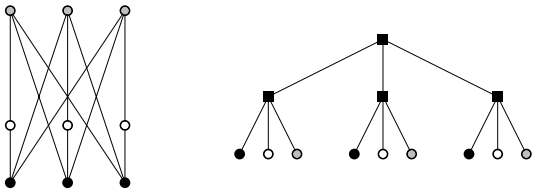


Figure 2 The graph obtained from $K_{3,3}$ by subdividing a matching belongs to $\mathcal{TM}_3(2)$. The respective tree model is depicted on the right.

Besides tree-width, another useful width measure of graphs is *clique-width*; defined for a graph G as the smallest number of labels $k = cw(G)$ such that G can be constructed using operations to create a new vertex with label i, take the disjoint union of two labeled graphs, add all edges between vertices of label i and label j, and relabel all vertices with label i to have label j. Similarly as tree-depth is related to tree-width, there exists a very new notion of *shrub-depth* [14] which is (in a sense) related to clique-width, and which we explain next.

▶ **Definition 2.2** (Tree model [14]). We say that a graph G has a *tree model of m colours and depth $d \geq 1$* if there exists a rooted tree T (of height d) such that

i. the set of leaves of T is exactly $V(G)$,
ii. the length of each root-to-leaf path in T is exactly d,
iii. each leaf of T is assigned one of m colours (this is not a graph colouring, though),
iv. and the existence of a G-edge between $u, v \in V(G)$ depends solely on the colours of u, v and the distance between u, v in T.

The class of all graphs having a tree model of m colours and depth d is denoted by $\mathcal{TM}_m(d)$.

For instance, $K_n \in \mathcal{TM}_1(1)$ or $K_{n,n} \in \mathcal{TM}_2(1)$. Definition 2.2 is further illustrated in Figure 2. It is easy to see that each class $\mathcal{TM}_m(d)$ is closed under complements and induced subgraphs, but neither under disjoint unions, nor under subgraphs. One can also routinely verify that each class $\mathcal{TM}_m(d)$ is of bounded clique-width. The depth d of a tree model can be seen as a generalization of the aforementioned tree-depth parameter, and for that reason it is useful to work with a more streamlined notion which only requires a single parameter. To this end we introduce the following (and we refer to [14] for additional details):

▶ **Definition 2.3** (Shrub-depth [14]). A class of graphs $\mathcal{G}$ has *shrub-depth d* if there exists m such that $\mathcal{G} \subseteq \mathcal{TM}_m(d)$, while for all natural m it is $\mathcal{G} \not\subseteq \mathcal{TM}_m(d-1)$.

Note that Definition 2.3 is asymptotic as it makes sense only for infinite graph classes. Particularly, classes of shrub-depth 1 are known as the graphs of bounded *neighbourhood diversity* in [17], i.e., those graph classes on which the twin relation on pairs of vertices (for a pair to share the same set of neighbours besides this pair) has a finite index.

MSO logic on graphs

Monadic second-order logic (MSO) is an extension of first-order logic (FO) by quantification over sets. On the one-sorted adjacency model of graphs it specifically reads as follows:

▶ **Definition 2.4** (MSO_1 logic of graphs). The language of MSO_1 contains the expressions built from the following elements:

- variables $x, y, \ldots$ for vertices, and $X, Y, \ldots$ for sets of vertices,

- the predicates $x \in X$ and $edge(x, y)$ with the standard meaning,
- equality for variables, the connectives $\wedge, \vee, \neg, \rightarrow$, and the quantifiers $\forall, \exists$ over vertex and vertex-set variables.

Note that we do not allow quantification over edges or sets of edges (as edges are not elements) in MSO_1. If we consider the two-sorted incidence graph model (in which the edges formed another sort of elements), then we get:

▶ **Definition 2.5** (MSO_2 logic of graphs)**.** The language of MSO_2 contains the expressions built from elements of MSO_1 plus the following:

- variables $e, f, \ldots$ for edges, $E, F, \ldots$ for sets of edges, the respective quantifiers, and
- the predicates $e \in F$ and $inc(x, e)$ with the standard meaning.

Already MSO_1 logic is quite powerful as it can express various common hard graph properties; e.g., 3-colourability. The expressive power of MSO_2 is even strictly larger [8] since, for instance, Hamiltonicity has an MSO_2 definition (while not MSO_1). On the other hand, MSO_2 and MSO_1 coincide on the class of trees, or on many other restricted graph classes. Hence we will speak only about MSO_1 on trees, from now on. The large expressive power of MSO logics is the reason for their popularity in algorithmic metatheorems.

The problem to decide, for a sentence ψ in logic $\mathcal{L}$, whether an input structure G satisfies $G \models \psi$, is also commonly called the $\mathcal{L}$ *model checking problem* (of ψ). Hence, for instance, the c-colourability problem for each fixed c is an instance of MSO_1 model checking; where $\psi \equiv \exists X_1, \ldots, X_c. \left[\left(\forall x. \bigvee_{i=1}^{c} x \in X_i \right) \wedge \left(\forall x, y. \bigwedge_{i=1}^{c} (x \notin X_i \vee y \notin X_i \vee \neg\, edge(x, y)) \right) \right]$.

3 Trees of Bounded Height and MSO

The primary purpose of this section is to prove Theorem 3.2; that for any m-coloured tree T of constant height h there exists an efficiently computable subtree $T_0 \subseteq T$ such that, for any MSO_1 sentence ϕ of fixed quantifier rank r, it is $T \models \phi \iff T_0 \models \phi$, and the size of T_0 is bounded by an elementary function of r and m (the dependence on h being non-elementary, though). Particularly, since checking of an MSO_1 property ϕ can be easily solved in time $\mathcal{O}^*\left(2^{c|\phi|}\right)$ on a graph with c vertices (in this case T_0) by recursive exhaustive expansion of all quantifiers of ϕ, this gives a kernelization-based elementary FPT algorithm for MSO_1 model checking of rooted m-coloured trees of constant height h (Corollary 3.3).

We need a bit more formal notation. The *height* h of a rooted tree T is the farthest distance from its root, and a node is at the *level* ℓ if its distance from the root is $h - \ell$. For a node v of a rooted tree T, we call a *limb of* v a subtree of T rooted at some child node of v. Our rooted trees are unordered, and they "grow top-down", i.e. we depict the root on the top. For this section we also switch from considering m-coloured trees to more convenient t-labelled ones, the difference being that one vertex may have several labels at once (and so $m \sim 2^t$). MSO_1 logic is naturally extended to labelled graphs by adding unary predicates $L(x)$ for every label L. We say that two such rooted labelled trees are *l-isomorphic* if there is an isomorphism between them preserving the root and all labels.

3.1 The Reduction Lemma

Concretely, we preprocess a given tree T into a bounded kernel $T_0 \subseteq T$ by recursively deleting from T all limbs which are "repeating (being l-isomorphic) too many times". This is formalized in Lemma 3.1. To describe the exact reduction of T to T_0, we need to define

the following recursive "threshold" values, for $i = 0, 1, 2, \ldots$:

$$R_i(q,s,k) = q \cdot N_i(q,s,k)^s, \qquad \text{where} \tag{1}$$

$$N_0(q,s,k) = 2^k + 1 \geq 2 \quad \text{and}$$
$$N_{i+1}(q,s,k) = 2^k \cdot \big(R_i(q,s,k)+1\big)^{N_i(q,s,k)} \leq 2^k \cdot \big(2q \cdot N_i(q,s,k)^s\big)^{N_i(q,s,k)} \tag{2}$$

For clarity, we informally in advance outline the intended meaning of these values R_i and N_i. We say a labelled rooted tree of height i is (q,s,k)-*reduced* if, at any level j, $0 < j \leq i$, each node of T has at most (1) $R_{j-1}(q,s,k)$ pairwise l-isomorphic limbs (which are of height $\leq j-1$). The value (2) $N_j(q,s,k)$ is then an upper bound on the number of all possible non-l-isomorphic rooted k-labelled trees T of height $\leq j$ that are (q,s,k)-reduced. Note that $N_0(q,s,k)$ accounts for all distinct k-labelled single-node trees and the empty tree.

Assume now any MSO_1 sentence (closed formula) ϕ with q element variables and s set variables, and height i. Then, provided $a, b \geq R_i(q,s,k)$ where $k = t + 3q + s$, we show that the sentence ϕ could not distinguish between a disjoint copies and b disjoint copies of any (q,s,k)-reduced rooted t-labelled tree of height i. Altogether formally:

▶ **Lemma 3.1.** *Let T be a rooted t-labelled tree of height h, and let ϕ be an MSO_1 sentence with q element quantifiers and s set quantifiers. Suppose that $u \in V(T)$ is a node at level $i+1$ where $i < h$.*

a) If, among all the limbs of u in T, there are more than $R_i(q,s,t+3q+s)$ pairwise l-isomorphic ones, then let $T' \subseteq T$ be obtained by deleting one of the latter limbs from T. Then, $T \models \phi \iff T' \models \phi$.

b) Consequently, there exists a rooted t-labelled tree $T_0 \subseteq T$ such that T_0 is $(q,s,t+3q+s)$-reduced, and $T \models \phi \iff T_0 \models \phi$.

In the case of FO logic, a statement analogous to Lemma 3.1 is obtained using folklore arguments of finite model theory (even full recursive expansion of all q vertex quantifiers in ϕ could "hit" only bounded number of limbs of u and the rest would not matter). However, in the case of MSO logic there are additional nontrivial complications which require new ideas (in addition to standard tools) in the proof. Briefly saying, one has to recursively consider the internal structure of the limbs of u, and show that even an expansion of a vertex-set quantifier in ϕ does not effectively distinguish too many of them (and hence some of them remain irrelevant for the decision whether $T \models \phi$).

Proof of Lemma 3.1. Note first that part b) readily follows by a recursive bottom-up application of a) to the whole tree. Hence we focus on a), and sketch our proof as follows:

(I) We are going to use a so called "quantifier elimination" approach.[1] That means, assuming $T \models \phi \not\iff T' \models \phi$, we look at the "distinguishing choice" of the first quantifier in ϕ, and encode it in the labeling of T (e.g., when $\phi \equiv \exists x.\psi$, we give new exclusive labels to the value of x and to its parent/children in T and T'). By an inductive assumption, we then argue that the shorter formula ψ cannot distinguish between these newly labeled T and T', which is a contradiction.

(II) The traditional quantifier elimination approach—namely of set quantifiers in ϕ, however, might not be directly applicable to even very many pairwise l-isomorphic limbs in T if their size is unbounded. Roughly explaining, the problem is that a single valuation of a set variable on these repeated limbs may potentially pairwise distinguish

[1] This approach has been inspired by recent [7], though here it is applied in a wider setting of MSO logic.

$$\begin{array}{ccc} \psi \in \text{“}\mathcal{L}_1 \text{ over } \mathscr{K}\text{”} & \xrightarrow{\ I\ } & \psi^I \in \text{“}\mathcal{L}_2 \text{ over } \mathscr{M}\text{”} \\ H \in \mathscr{K} & & G \in \mathscr{M} \\ \\ G^I \cong H & \xleftarrow{\ I\ } & G \\ (\text{s.t. } G^I \models \psi) & & (\text{s.t. } G \models \psi^I) \end{array}$$

Figure 3 A basic informal scheme of an interpretation of $\mathrm{Th}_{\mathcal{L}_1}(\mathscr{K})$ into $\mathrm{Th}_{\mathcal{L}_2}(\mathscr{M})$.

all of them. Hence additional combinatorial arguments are necessary to bound the size of the limbs in consideration.

(III) Having successfully resolved technical (II), the rest of the proof is a careful composition of inductive arguments using the formula (1) $R_i(q,s,k) = q \cdot N_i(q,s,k)^s$.

Details can be found in the full paper [12]. ◀

3.2 Algorithmic applications

With some calculus, we summarize the obtained result from an algorithmic point of view. Let $exp^{(i)}(x)$ be the i-fold exponential function defined inductively as follows: $exp^{(0)}(x) = x$ and $exp^{(i+1)}(x) = 2^{exp^{(i)}(x)}$. Note that $exp^{(h)}(x)$ is an elementary function of x for each particular height h. For a rooted t-labelled tree T of height $\leq h$, we call the uniquely-determined maximal (q,s,k)-reduced tree $T_0 \subseteq T$ from Lemma 3.1 b), where $k = t + 3q + s$, a (q,s,k)-*reduction* of the tree T. Then we routinely get:

▶ **Theorem 3.2.** *Let $t, h \geq 1$ be integers, and let ϕ be an MSO_1 sentence with q element quantifiers and s set quantifiers. For each rooted t-labelled tree T of height h, the tree $T_0 \subseteq T$ which is a $(q, s, t + 3q + s)$-reduction of T and $T_0 \models \phi \iff T \models \phi$, can be computed in linear time (non-parameterized) from T. Moreover, its size is bounded by*

$$|V(T_0)| \leq exp^{(h)}\left[(2^{h+5} - 12) \cdot (t + q + s)(q + s)\right].$$

▶ **Corollary 3.3.** *Let T be a rooted t-labelled tree of constant height $h \geq 1$, and let ϕ be an MSO_1 sentence with r quantifiers. Then $T \models \phi$ can be decided by an FPT algorithm in time*

$$\mathcal{O}\left(exp^{(h+1)}\left[2^{h+5} \cdot r(t+r)\right] + |V(T)|\right) = \mathcal{O}\left(exp^{(h+1)}(|\phi|^2) + |V(T)|\right).$$

The arguments of Corollary 3.3 can be further extended to suitable classes of general graphs via the traditional tool of *interpretability* of logic theories [20]. This powerful tool, however, has rather long formal description, and since we are going to use it only ad hoc in some proofs anyway, we provide here only a brief conceptual sketch. Imagine two classes of relational structures $\mathscr{K}, \mathscr{M}$ and two logical languages $\mathcal{L}_1, \mathcal{L}_2$. We say there is an *interpretation* I of the $\mathcal{L}_1$ theory of $\mathscr{K}$ into the $\mathcal{L}_2$ theory of $\mathscr{M}$ if (see Figure 3)

- there exist $\mathcal{L}_2$ formulas which can “define” the domain and the relations of each structure $H \in \mathscr{K}$ inside a suitable structure $G \in \mathscr{M}$, formally $H \simeq G^I$,
- and each formula $\psi \in \mathcal{L}_1$ over $\mathscr{K}$ can be accordingly translated into $\psi^I \in \mathcal{L}_2$ over $\mathscr{M}$ such that “truth is preserved”, i.e., $H \models \psi$ iff $G \models \psi^I$ for all such related H, G.

A simple example is an interpretation of the complement of a graph G into G itself via defining the edge relation as $\neg\, edge(x,y)$. A bit more complex example is shown by interpreting a line graph $L(G)$ of a graph G inside G; the domain (vertex set) of $L(G)$ being interpreted in $E(G)$, and the adjacency relation of $L(G)$ defined by the formula $\alpha(e,f) \equiv$

$e \neq f \wedge \exists x.\ inc(x,e) \wedge inc(x,f)$. This example interprets the MSO_1 theory of line graphs in the MSO_2 theory of graphs.

We now return back to the promised extensions. Since the MSO_2 theory of graphs of tree-depth $\leq d$ has an interpretation in coloured trees of depth $\leq d+1$ (a graph G is actually interpreted in W such that $G \subseteq cl(W)$, with labels determining which "back edges" of W belong to G), we get the following generalization of Lampis' [17] from Corollary 3.3: MSO_2 model checking can be done in FPT time which depends elementarily on the checked formula, not only for graphs of bounded vertex cover, but also for those of bounded tree-depth.

▶ **Theorem 3.4.** *Let $\mathscr{D}_d$ denote the class of all graphs of tree-depth $\leq d$, and ϕ be an* MSO_2 *sentence with r quantifiers. Then the problem of deciding $G \models \phi$ for $G \in \mathscr{D}_d$ has an FPT algorithm with runtime $\mathcal{O}\big(\, exp^{(d+2)}\big(2^{3d+7} \cdot r^2\big) + |V(G)|\big)$.*

We also remark on an important aspect of FPT algorithms using width parameters—how to *obtain the associated decomposition* of the input (here of $G \in \mathscr{D}_d$). In the particular case of tree-depth, the answer is rather easy since one can use the linear FPT algorithm for tree-decomposition [2] to compute it (while, say, for clique-width this is an open problem).

Concerning MSO_1 model checking, one can go further. Graphs of *neighbourhood diversity* m (introduced in [17]) are precisely those having a model in which every vertex receives one of m colours, and the existence of an edge between u, v depends solely on the colours of u, v. Clearly, these graphs coincide with those having a tree model of m colours and depth 1, and so we can give an FPT algorithm for MSO_1 model checking on them from Corollary 3.3, which is an alternative derivation for another result of Lampis [17]. We can similarly derive an estimation of the main result of [13] (here just one exponential fold worse).

A common generalization of these particular applications of Corollary 3.3 has been found, together with the new notion of shrub-depth, in this subsequent work:

▶ **Theorem 3.5** (Ganian et al. [14])**.** *Assume $d \geq 1$ is a fixed integer. Let $\mathscr{G}$ be any graph class of shrub-depth d (Definition 2.3). Then the problem of deciding $G \models \phi$ for the input $G \in \mathscr{G}$ and* MSO_1 *sentence ϕ, can be solved by an FPT algorithm, the runtime of which has an elementary dependence on the parameter ϕ. This assumes G is given on the input alongside with its tree model of depth d.*

4 Expressive power of FO and MSO

Theorem 3.2 has another interesting corollary in the logic domain. Since the size of the reduction T_0 of T is bounded independently of T, the outcome of $T \models \phi$ actually depends on a finite number of fixed-size cases, and one can use even FO logic to express (one would say by brute force) which of these cases is the correct $(q, s, t + 3q + s)$-reduction of T. The outlined arguments lead to the following conclusions.

▶ **Theorem 4.1** (Theorem 3.2)**.** *Let $t, h \geq 1$ be integers, and let ϕ be an* MSO_1 *sentence with q element quantifiers and s set quantifiers. There exists a finite set of rooted t-labelled trees $\mathscr{U}_{h,t,\phi}$ satisfying the following: For any rooted t-labelled tree T of height $\leq h$, it holds $T \models \phi$ if and only if the $(q, s, t + 3q + s)$-reduction of T is l-isomorphic to a member of $\mathscr{U}_{h,t,\phi}$.*

With Theorem 4.1 we get quite close to the very recent achievement of Elberfeld, Grohe, and Tantau [9] who prove that FO and MSO_2 have equal expressive power on the graphs of bounded tree-depth (and that this condition is also necessary on hereditary graph classes). The following weaker statement is actually an easy consequence of our findings, too:

▶ **Corollary 4.2** (Elberfeld, Grohe, and Tantau [9]). *Let h, t be integers, and ϕ an* MSO_1 *sentence. Then there is an FO sentence $\psi_{h,t,\phi}$ such that, for any rooted t-labelled tree T of height $\leq h$, it is $T \models \phi \iff T \models \psi_{h,t,\phi}$.*

It is now a natural question whether and how could our alternative approach to coincidence between FO and MSO on graphs be extended in the same direction.

Indeed, given an MSO_2 sentence ϕ over $\mathscr{D}_d$ (the graphs of tree-depth $\leq d$), we can interpret this in an MSO_1 sentence ϕ_d^I over rooted $(d+1)$-labelled trees of height $\leq d+1$. Then, by Corollary 4.2, we immediately get an FO sentence σ_d equivalent to ϕ_d^I. The problem is, however, that σ_d is a formula over rooted $(d+1)$-labelled trees, and we would like to get an interpretation of σ_d back in the FO theory of the class $\mathscr{D}_d$, which does not seem to be an easy task directly. Still, part of the arguments of [9] can be combined with the approach of Corollary 4.2 to provide an alternative relatively short proof of coincidence between FO and MSO_1 on classes of bounded tree-depth (thus bypassing the Feferman–Vaught–type theorem in [9]).

The reason for specifically mentioning Elberfeld, Grohe, and Tantau's [9] here is actually their main posted question—what are the sufficient and necessary conditions for a hereditary graph class to guarantee the same expressive power of FO and MSO_1? Using Theorem 4.1 and improved ideas based on a proof of Corollary 4.2, we provide a nontrivial sufficient condition which we also conjecture to be necessary.

▶ **Theorem 4.3.** *Let d be an integer and $\mathscr{S}$ be any graph class of shrub-depth d (Definition 2.3). Then for every* MSO_1 *sentence ϕ there is an* FO *sentence $\psi_{d,\phi}$ such that, for any $G \in \mathscr{S}$, it is $G \models \phi \iff G \models \psi_{d,\phi}$. Consequently,* FO *and* MSO_1 *have the same expressive power on $\mathscr{S}$.*

▶ Conjecture 4.4. Consider a hereditary (i.e., closed under induced subgraphs) graph class $\mathscr{S}$. If the expressive powers of FO and MSO_1 are equal on $\mathscr{S}$, then the shrub-depth of $\mathscr{S}$ is bounded (by a suitable constant).

The key to proving Theorem 4.3 is the notion of twin sets. Recall that two vertices $x, y \in V(G)$ are called *twins* if their neighbour sets in $G - x - y$ coincide. Though the edge xy is not specified in this definition, it easily follows that whenever we have a set of pairwise twins in G, then those induce a clique or an independent set.

▶ **Definition 4.5** (Twin sets). Assume $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_k\}$ are disjoint indexed sets (k-tuples) of vertices of a graph G. We say that X, Y are *twin sets* in G if

- the subgraphs of G induced on X and on Y are identical, i.e., $x_i x_j \in E(G)$ iff $y_i y_j \in E(G)$ for all pairs $i, j \in \{1, \ldots, k\}$, and
- for $i = 1, \ldots, k$, the set of neighbours of x_i in $V(G) \setminus (X \cup Y)$ equals that of y_i.

Note that, for simplicity, we consider the twin-sets relation only on disjoint sets, and that this relation is generally not transitive. Although we do not need more for this paper, we suggest that the notion deserves further extended study elsewhere.

A tree model (Definition 2.2) of a graph G can be, informally, viewed as a complete recursive decomposition (of bounded depth) of G into groups of pairwise disjoint pairwise twin sets. Roughly, an application of Lemma 3.1 a) then says that if (at any level) the number of pairwise twin sets in a group is "too large", then one of these sets can be deleted from G without affecting validity of a fixed MSO_1 property on G. Our main task is to describe "reducibility" of a large group of twin sets in G using FO (the sets having bounded size, though), which is more complicated than in the tree-depth case due to lack of some "nice connectivity properties" of a tree-depth decomposition.

Proof outline (Theorem 4.3). We assume a graph $G \in \mathscr{S}$ with a tree model T of constant depth d, and an MSO_1 sentence ϕ. We informally continue as follows.

(I) For every fixed d, one can easily interpret ϕ in an MSO_1 formula ϕ_d^I over T, such that $G \models \phi \iff T \models \phi_d^I$.

(II) By Definition 2.2, pairwise l-isomorphic sibling limbs in T correspond to a group of pairwise twin sets in G. Deleting one of these sets from G is equivalent to deleting the corresponding limb from T. Hence by (I) and Theorem 4.1, there is a finite set $\mathscr{U}_\phi$ of graphs (independent of G) such that $G \models \phi$ iff G "reduces" to a member of $\mathscr{U}_\phi$.

(III) The meaning of "reduction" is analogous to Section 3.1, to a (q,s,k)-reduced subtree of the tree model T. The minor technical differences are; (1) we can describe the reduction using twin sets, without an explicit reference to whole T, and (2) we actually aim at a $(q,s,k)'$-*reduction* which means the reduction threshold values are $R'_j(q,s,k) = \max\{R_j(q,s,k), 2\}$. (We need to guarantee that at least two twin sets of each group remain after the reduction, even in degenerate cases.)

(IV) We provide an FO definition of the fact that G reduces to $H \in \mathscr{U}_\phi$, modulo some technical details. This FO formula ϱ_H depends mainly on d and H (actually on a suitable tree model of H). The desired sentence $\psi_{d,\phi}$ in Theorem 4.3 is then constructed as the (*finite*) disjunction $\psi_{d,\phi} \equiv \bigvee_{H \in \mathscr{U}_\phi} \varrho_H$.

Now we give the crucial technical detail and the related claims which make step (IV) working. Assume T is a tree model of a graph G, and B is a limb of a node v in T, such that W is the set of leaves of B. We say that a tree model T' is obtained from T by *splitting* B *along* $X \subseteq W$ if a disjoint copy B' of B with the same parent v is added into T, and then B is restricted to a rooted Steiner tree of $W \setminus X$ while B' is restricted to a rooted Steiner tree of X' (the corresponding copy of X). A tree model T is *splittable* if some limb in T can be split along some subset X of its leaves, making a tree model T' which represents the same graph G as T does. A tree model is *unsplittable* if it is not splittable. Notice that any tree model can be turned into an unsplittable one; simply since the splitting process must end eventually.

▶ **Lemma 4.6.** *Let H be a graph, and $R \subseteq H$ be an induced subgraph having a tree model T (of m colours and depth d, but this is not relevant). Let T contain two disjoint l-isomorphic limbs B, B' of a node v, and a limb C of a node u. The position of C against B, B' can be arbitrary (it may be $u = v$ or even $C = B$ or $C = B'$), as long as C is disjoint from one of B, B'. Let $W, W' \subseteq V(R)$ denote the sets of leaves of B, B', respectively, and $X \subseteq V(R)$ denote the set of leaves of C. Assume $Y, Z \subseteq V(H) \setminus V(R)$ are such that W, W', Y are pairwise twin sets in H, and that X, Z are also twin sets in H. If $Y \neq Y \cap Z \neq \emptyset$, then the tree model T of R is* splittable.

▶ **Lemma 4.7.** *Let $m, d \geq 1$ and q, s be integers. Assume $G \in \mathcal{TM}_m(d)$ is a graph, and $R \subseteq G$ is an induced subgraph having an* unsplittable *tree model T (of m colours and depth d). Let $\widehat{x_R} = (x_v : v \in V(R))$ be a vector of free variables valued in the respective vertices of R in G. Then there exists an* FO *formula ϱ_T, depending on d, m, q, s, and T, such that the following holds: $G \models \varrho_T(\widehat{x_R})$ if, and only if, $R \subseteq G$ and there exists a tree model $T' \supseteq T$ of G of m colours and depth d, such that the $(q, s, m + 3q + s)'$-reduction of T' is T.*

The importance of Lemma 4.6 in the proof of Lemma 4.7 is, informally, in that one can focus just on including every vertex of $G - R$ into some set which is twin (possibly after recursive reduction) to suitable limbs of T, while such sets will then never overlap. See Figure 4. With Lemma 4.7 at hand, it is then straightforward (though technical and not

.

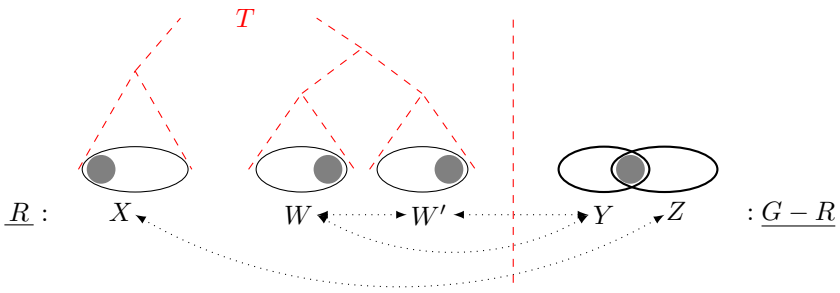


Figure 4 A situation which cannot happen, in a graph G with an unsplittable tree model T of an induced subgraph $R \subseteq G$, and with the sets W, W', Y and X, Z as in Lemma 4.6.

short) to finish the proof of Theorem 4.3 along the aforementioned outline. Details can be found in the full paper [12]. ◀

5 Conclusions

We briefly recapitulate the two-fold contribution of our primary result; that the MSO model checking problem on the universe of coloured trees of bounded height can be reduced to a kernel of size bounded by an elementary function of the formula. Firstly, it allows us to easily obtain nontrivial extensions of Lampis' and Ganian's result and to fill the gap set by Courcelle's theorem and the negative result of Frick and Grohe.

Secondly, it provides an alternative simple and intuitive way of understanding of *why* on some classes of graphs FO and MSO logics coincide. In this respect, the most important property of our kernel is that, after seeing more than a certain number of copies of a certain substructure in the input graph, the validity of an MSO formula in question does not change any further. While such a behavior is natural for FO properties, it is somehow surprising to see it for much wider MSO. This "loss of expressiveness" of MSO (getting down to the FO level) is inherited by graph classes of bounded tree-depth and shrub-depth.

Finally, we briefly discuss why we believe Conjecture 4.4 holds true. It is known [9] that each subgraph closed class of graphs such that $\mathrm{FO} = \mathrm{MSO}_2$ has to have bounded tree-depth. Both classes of bounded tree-depth and classes of bounded shrub-depth are interpretable in trees of bounded depth, the main difference is how "dense" they are. By allowing "too many" edges in graphs of bounded shrub-depth, we basically lost the ability to address edges of the interpreted graph in the underlying tree and hence also the ability to quantify over these edges and sets of edges (notice that this also means that our class of graphs is no longer closed under taking subgraphs, but is still hereditary). Since this is exactly the difference between MSO_1 and MSO_2, classes of graphs of bounded shrub-depth are natural candidates for exactly those hereditary classes where $\mathrm{FO} = \mathrm{MSO}_1$.

References

1 S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

2 H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

3 B. Courcelle. The monadic second order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85:12–75, 1990.

4 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

5 R. Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, New York, 2005.

6 R. Downey and M. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer, 1999.

7 Z. Dvořák, D. Král', and R. Thomas. Deciding first-order properties for sparse graphs. In *FOCS*, pages 133–142, 2010.

8 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1999.

9 M. Elberfeld, M. Grohe, and T. Tantau. Where first-order and monadic second-order logic coincide. In *LICS*, pages 265–274, 2012.

10 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

11 M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

12 J. Gajarský and P. Hliněný. Faster deciding mso properties of trees of fixed height, and some consequences. *arXiv*, abs/1204.5194, 2012.

13 R. Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In *IPEC'11*, volume 7112 of *LNCS*, pages 259–271. Springer, 2012.

14 R. Ganian, P. Hlinený, J. Nesetril, J. Obdrzálek, P. O. de Mendez, and R. Ramadurai. When trees grow low: Shrubs and fast mso1. In B. Rovan, V. Sassone, and P. Widmayer, editors, *MFCS*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012.

15 M. Grohe and S. Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics: AMS-ASL Special Session, January 5-8, 2009*, Contemporary Mathematics, pages 181–206. Amer. Mathematical Society, 2011.

16 S. Kreutzer. Algorithmic meta-theorems. *Electronic Colloquium on Computational Complexity (ECCC)*, TR09-147, 2009.

17 M. Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, Sept. 2012.

18 J. Nešetřil and P. Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European J. Combin.*, 27(6):1024–1041, 2006.

19 J. Nešetřil and P. Ossona de Mendez. *Sparsity (Graphs, Structures, and Algorithms)*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012. 465 pages.

20 M. O. Rabin. A simple method for undecidability proofs and some applications. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Sciences*, volume 1, pages 58–68. North-Holland, Amsterdam, 1964.

21 M. O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, July 1969.

Cost-Parity and Cost-Streett Games*

Nathanaël Fijalkow[1,2] and Martin Zimmermann[2]

1 **LIAFA, Université Paris 7.** nath@liafa.univ-paris-diderot.fr
2 **Institute of Informatics, University of Warsaw.** zimmermann@mimuw.edu.pl

Abstract

We consider two-player games played on finite graphs equipped with costs on edges and introduce two winning conditions, cost-parity and cost-Streett, which require bounds on the cost between requests and their responses. Both conditions generalize the corresponding classical ω-regular conditions as well as the corresponding finitary conditions.

For cost-parity games we show that the first player has positional winning strategies and that determining the winner lies in $\mathbf{NP} \cap \mathbf{coNP}$. For cost-Streett games we show that the first player has finite-state winning strategies and that determining the winner is **EXPTIME**-complete. This unifies the complexity results for the classical and finitary variants of these games. Both types of cost games can be solved by solving linearly many instances of their classical variants.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Parity Games, Streett Games, Costs, Scoring Functions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.124

1 Introduction

In recent years, boundedness problems arose in topics pertaining to automata and logics leading to the development of novel models and techniques to tackle these problems. Although in general undecidable, many boundedness problems for automata turn out to be decidable if the acceptance condition can refer to boundedness properties of variables, but the transitions cannot access variable values. A great achievement was made by Hashiguchi [15] who proved decidability of the star-height problem by reducing it to a boundedness problem for a certain type of finite automaton and by solving this problem. This led the path to recent developments towards a general theory of bounds in automata and logics, comprising automata and logics with bounds [1, 3], satisfiability algorithms for these logics [2, 4], and regular cost-functions [10].

In this work, we consider boundedness problems in turn-based two-player graph games of infinite duration. Using the acceptance condition of the automata models of [3] (namely ωB- and ωS automata) yields games that are equivalent to ω-regular games. Hence, we take a different route and introduce cost-parity and cost-Streett conditions which generalize the (classical) ω-regular parity- respectively Streett condition, as well as the finitary parity- respectively finitary Streett condition [8]. While both finitary variants strengthen the classical conditions by adding bounds, the complexity of solving these games diverges: (in the state of the art) finitary parity games are simpler than parity games, while finitary Streett games are harder than Streett games. Indeed solving finitary parity games can be carried out in polynomial time [8], while no polynomial-time algorithm for parity games is yet known, and

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454 (GALE) and n° 239850 (SOSNA).

the decision problem is in **NP** ∩ **coNP**. The situation is reversed for Streett games, since solving them is **coNP**-complete [13] while solving finitary Streett games is **EXPTIME**-complete. The latter result is shown in unpublished work by Chatterjee, Henzinger, and Horn: by slightly modifying the proof of **EXPTIME**-hardness of solving request-response games presented in [9] they prove **EXPTIME**-hardness of solving finitary Streett games.

A cost-parity game is played on an arena whose vertices are colored by natural numbers, and where traversing an edge incurs a non-negative cost. Player 0 wins a play if there is a bound b such that almost all odd colors (which we think of as requests) are followed by a larger even color (which we think of as responses) that is reached with cost at most b. The definition of (cost-) Streett games goes along the same lines, but the requests and responses are independent and not hierarchically ordered as in parity conditions. The cost of traversing an edge can be used to model the consumption of a resource. Thus, if Player 0 wins a play she can achieve her goal along an infinite run with bounded resources. On the other hand, Player 1's objective is to exhaust the resource, no matter how big the capacity is.

We show that cost-parity games enjoy two nice properties of parity and finitary parity games: Player 0 has memoryless winning strategies, and determining the winner lies in **NP** ∩ **coNP**. Furthermore, we show that solving cost-parity games can be algorithmically reduced to solving parity games, which allows to solve these games almost as efficiently as parity games. We then consider cost-Streett games and prove that Player 0 has finite-state winning strategies, and that determining the winner is **EXPTIME**-complete. Our complexity results unify the previous results about finitary parity and Streett games and the results about their classical variants.

To obtain our results, we present an algorithm to solve cost-parity games that iteratively computes the winning region of Player 0 employing an algorithm to solve parity games. As a by-product, we obtain finite-state winning strategies for Player 0. We improve this by showing how to transform a finite-state winning strategy into a positional winning strategy. This construction relies on so-called scoring functions (which are reminiscent of the simulation of alternating tree-automata by nondeterministic automata presented in [18]) and presents a general framework to turn finite-state strategies into positional ones, which we believe to be applicable in other situations as well. Finally, we present an algorithm that solves cost-Streett games by solving Streett games. Here, we show the existence of finite-state winning strategies for Player 0 in cost-Streett games.

In our proof, we reduce games with boundedness winning conditions to games with ω-regular winning conditions. This is reminiscent of the solution of the domination problem for regular cost-functions on finite trees [11]. In contrast to this work, which is concerned with proving decidability, we are interested in efficient algorithms. Hence, we need a more sophisticated reduction and a careful analysis of the memory requirements.

Adding quantitative requirements to qualitative winning conditions has been an active field of research during the last decade. Just recently, there has been a lot of interest in so-called energy games, whose winning conditions are boundedness requirements on the consumption of resources. Solving energy games with multiple resources is in general intractable [14] while so-called consumption games, a subclass of energy games, are shown to be tractable in [5]. Furthermore, energy parity games, whose winning conditions are a conjunction of an (single resource) energy and a parity condition, can be solved in **NP** ∩ **coNP** and Player 0 has positional winning strategies [7]. Although the first two results are similar to our results on cost-parity games, the energy parity condition does not relate the energy consumption to the parity condition. In contrast, the costs in cost-parity games give a qualitative measure of the satisfaction of the parity condition.

2 Definitions

Infinite Games. A (cost) arena $\mathcal{A} = (V, V_0, V_1, E, \mathrm{Cst})$ consists of a finite, directed graph (V, E), a partition $\{V_0, V_1\}$ of V, and an edge-labeling $\mathrm{Cst}\colon E \to \{\varepsilon, i\}$. An edge with label i is called increment-edge, edges labeled by ε are called accordingly ε-edges. We extend the edge-labeling to a cost function over finite and infinite paths obtained by counting the number of increment-edges traversed along the path. A play in $\mathcal{A}$ starting in $v \in V$ is an infinite path $\rho = \rho_0\rho_1\rho_2\cdots$ such that $\rho_0 = v$. To avoid the nuisance of dealing with finite plays, we assume every vertex to have an outgoing edge.

A game $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ consists of an arena $\mathcal{A}$ and a set $\mathrm{Win} \subseteq V^\omega$ of winning plays for Player 0. The set of winning plays for Player 1 is $V^\omega \setminus \mathrm{Win}$. A strategy for Player i is a mapping $\sigma\colon V^*V_i \to V$ such that $(v, \sigma(wv)) \in E$ for all $wv \in V^*V_i$. We say that σ is positional if $\sigma(wv) = \sigma(v)$ for every $wv \in V^*V_i$. A play $\rho_0\rho_1\rho_2\ldots$ is consistent with σ if $\rho_{n+1} = \sigma(\rho_0\cdots\rho_n)$ for every n with $\rho_n \in V_i$. A strategy σ for Player i is a winning strategy from a set of vertices $W \subseteq V$ if every play that starts in some $v \in W$ and is consistent with σ is won by Player i. The winning region $W_i(\mathcal{G})$ of Player i in $\mathcal{G}$ is the set of vertices from which Player i has a winning strategy. We say that a strategy is uniform, if it is winning from all $v \in W_i(\mathcal{G})$. We always have $W_0(\mathcal{G}) \cap W_1(\mathcal{G}) = \emptyset$. On the other hand, if $W_0(\mathcal{G}) \cup W_1(\mathcal{G}) = V$, then we say that $\mathcal{G}$ is determined. All games we consider in this work are determined. Solving a game amounts to determining its winning regions.

A memory structure $\mathcal{M} = (M, \mathrm{Init}, \mathrm{Upd})$ for an arena $(V, V_0, V_1, E, \mathrm{Cst})$ consists of a finite set M of memory states, an initialization function $\mathrm{Init}\colon V \to M$, and an update function $\mathrm{Upd}\colon M \times V \to M$. The update function can be extended to $\mathrm{Upd}^*\colon V^+ \to M$ in the usual way: $\mathrm{Upd}^*(\rho_0) = \mathrm{Init}(\rho_0)$ and $\mathrm{Upd}^*(\rho_0\ldots\rho_n\rho_{n+1}) = \mathrm{Upd}(\mathrm{Upd}^*(\rho_0\ldots\rho_n), \rho_{n+1})$. A next-move function (for Player i) $\mathrm{Nxt}\colon V_i \times M \to V$ has to satisfy $(v, \mathrm{Nxt}(v, m)) \in E$ for all $v \in V_i$ and all $m \in M$. It induces a strategy σ for Player i with memory $\mathcal{M}$ via $\sigma(\rho_0\ldots\rho_n) = \mathrm{Nxt}(\rho_n, \mathrm{Upd}^*(\rho_0\ldots\rho_n))$. A strategy is called finite-state if it can be implemented by a memory structure.

An arena $\mathcal{A} = (V, V_0, V_1, E, \mathrm{Cst})$ and a memory structure $\mathcal{M} = (M, \mathrm{Init}, \mathrm{Upd})$ for $\mathcal{A}$ induce the expanded arena $\mathcal{A} \times \mathcal{M} = (V \times M, V_0 \times M, V_1 \times M, E', \mathrm{Cst}')$ where $((v, m), (v', m')) \in E'$ if and only if $(v, v') \in E$ and $\mathrm{Upd}(m, v') = m'$, and $\mathrm{Cst}'((v, m)(v', m')) = \mathrm{Cst}(v, v')$. Every play ρ in $\mathcal{A}$ has a unique extended play $\rho' = (\rho_0, m_0)(\rho_1, m_1)(\rho_2, m_2)\ldots$ in $\mathcal{A} \times \mathcal{M}$ defined by $m_0 = \mathrm{Init}(\rho_0)$ and $m_{n+1} = \mathrm{Upd}(m_n, \rho_{n+1})$, i.e., $m_n = \mathrm{Upd}^*(\rho_0\cdots\rho_n)$. Note that every infix of ρ and the corresponding infix of ρ' have the same cost.

A game $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ is reducible to $\mathcal{G}' = (\mathcal{A}', \mathrm{Win}')$ via $\mathcal{M}$, written $\mathcal{G} \leq_{\mathcal{M}} \mathcal{G}'$, if $\mathcal{A}' = \mathcal{A} \times \mathcal{M}$ and every play ρ in $\mathcal{G}$ is won by the player who wins the extended play ρ' in $\mathcal{G}'$, i.e., $\rho \in \mathrm{Win}$ if and only if $\rho' \in \mathrm{Win}'$.

▶ **Lemma 1.** *Let $\mathcal{G}$ be a game with vertex set V and $W \subseteq V$. If $\mathcal{G} \leq_{\mathcal{M}} \mathcal{G}'$ and Player i has a finite-state winning strategy for $\mathcal{G}'$ from $\{(v, \mathrm{Init}(v)) \mid v \in W\}$, then she also has a finite-state winning strategy for $\mathcal{G}$ from W.*

Let $\mathcal{A} = (V, V_0, V_1, E, \mathrm{Cst})$ and let $i \in \{0, 1\}$. The i-attractor of $F \subseteq V$ in $\mathcal{A}$, denoted by $\mathrm{Attr}_i^{\mathcal{A}}(F)$, is the set of vertices from which Player i has a strategy such that every play that starts in this set and is consistent with the strategy visits F.

Cost-Parity Games. Let $\mathcal{A} = (V, V_0, V_1, E, \mathrm{Cst})$ be an arena and let $\Omega\colon V \to \mathbb{N}$ be a coloring of its vertices. In all games we are about to define, we interpret the occurrence of a color as request, which has to be answered by visiting a vertex of larger even color at a later position. By imposing conditions on the responses, we obtain three different types of

winning conditions. To simplify our notation, let $\mathrm{Ans}(c) = \{c' \in \mathbb{N} \mid c' \geq c \text{ and } c' \text{ even}\}$ be the set of colors that answer a request of color c. Note that $\mathrm{Ans}(c) \subseteq \mathrm{Ans}(c')$ for $c \geq c'$ and $c \in \mathrm{Ans}(c)$ if c is even.

The parity winning condition, denoted by $\mathrm{Parity}(\Omega)$, requires that almost all requests are eventually answered. Equivalently, $\rho \in \mathrm{Parity}(\Omega)$ if and only if the maximal color that occurs infinitely often in ρ is even. The finitary parity condition [8] is a strengthening of the parity condition; it requires the existence of a bound b such that almost all requests are answered within b steps. Formally, given a play $\rho = \rho_0\rho_1\rho_2\cdots$ and $k \in \mathbb{N}$, we define

$$\mathrm{Dist}(\rho, k) = \inf\{k' - k \mid k' \geq k \text{ and } \Omega(\rho_{k'}) \in \mathrm{Ans}(\Omega(\rho_k))\}$$

where we use $\inf \emptyset = \infty$. Hence, $\mathrm{Dist}(\rho, k)$ is the number of steps between the request at position k and its earliest response. The finitary parity winning condition is $\mathrm{FinParity}(\Omega) = \{\rho \in V^\omega \mid \limsup_{k\to\infty} \mathrm{Dist}(\rho, k) < \infty\}$.

Note that both the parity and the finitary parity condition do not depend on the cost function. Finally, by not bounding the number of steps between the requests and their responses, but by bounding the *cost* between requests and responses, we obtain the cost-parity condition. Given a play $\rho = \rho_0\rho_1\rho_2\cdots$ and $k \in \mathbb{N}$, we define the cost-of-response by

$$\mathrm{Cor}(\rho, k) = \inf\{\mathrm{Cst}(\rho_k \cdots \rho_{k'}) \mid k' \geq k \text{ and } \Omega(\rho_{k'}) \in \mathrm{Ans}(\Omega(\rho_k))\} \ .$$

The cost-parity winning condition is $\mathrm{CostParity}(\Omega) = \{\rho \in V^\omega \mid \limsup_{k\to\infty} \mathrm{Cor}(\rho, k) < \infty\}$.

▶ Remark 2. $\mathrm{FinParity}(\Omega) \subseteq \mathrm{CostParity}(\Omega) \subseteq \mathrm{Parity}(\Omega)$.

A game $\mathcal{G} = (\mathcal{A}, \mathrm{CostParity}(\Omega))$ is called cost-parity game. In a similar way, we define parity and finitary parity games. Note that if $\mathcal{A}$ contains no increment-edges, then we have $\mathrm{CostParity}(\Omega) = \mathrm{Parity}(\Omega)$, and if $\mathcal{A}$ contains no ε-edges, then $\mathrm{CostParity}(\Omega) = \mathrm{FinParity}(\Omega)$. Hence, cost-parity games generalize both parity and finitary parity games.

Since cost-parity conditions can be shown to be on the third level of the Borel hierarchy, we obtain the following result as a consequence of the Borel determinacy theorem [17].

▶ Remark 3. Cost-parity games are determined.

Fix a play ρ. We say that a request at position k is answered with cost c, if $\mathrm{Cor}(\rho, k) = c$. Note that a request at a position k with an even color is answered with cost 0. Furthermore, we say that a request at position k is unanswered with cost c, if there is no position $k' \geq k$ such that $\Omega(\rho_{k'}) \in \mathrm{Ans}(\Omega(\rho_k))$, but we have $\mathrm{Cst}(\rho_k\rho_{k+1}\cdots) = c$, i.e., there are exactly c increment-edges after position k. Finally, we say that a request at position k is unanswered with cost ∞, if there is no position $k' \geq k$ such that $\Omega(\rho_{k'}) \in \mathrm{Ans}(\Omega(\rho_k))$ and we have $\mathrm{Cst}(\rho_k\rho_{k+1}\cdots) = \infty$, i.e., there are infinitely many increment-edges after position k. We say that a request is unanswered, if it is unanswered with cost in $\mathbb{N} \cup \{\infty\}$. We often use the following equivalence.

▶ Remark 4. A play ρ has only finitely many unanswered requests if and only if $\rho \in \mathrm{Parity}(\Omega)$.

▶ **Example 5.** Consider the cost-parity game depicted in Figure 1 where all vertices belong to V_1, and the label of a vertex denotes its name (in the upper part) and its color (in the lower part). Player 1 wins from $\{a, b, c\}$ by requesting color 1 at vertex a infinitely often and staying at vertex b longer and longer, but also visiting c infinitely often (and thereby answering the request). Note that this strategy is not finite-state. Indeed, one can easily prove that Player 1 does not have a finite-state winning strategy for this game. Player 0 wins from every other vertex, since Player 1 can raise only finitely many requests from these vertices, albeit these requests are unanswered with cost ∞.

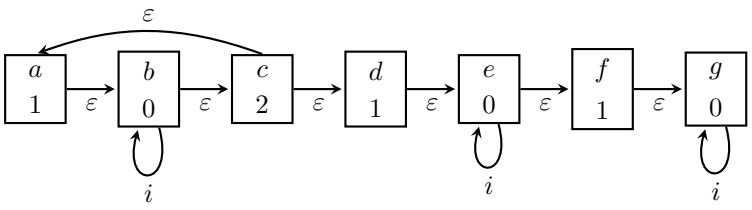


Figure 1 A cost-parity game.

3 Solving Cost-Parity Games

In this section, we show how to determine the winning regions in a cost-parity game. Our algorithm proceeds in two steps: in Subsection 3.1, we show that it suffices to solve games whose winning condition is a strengthening of the cost-parity condition, the so-called bounded cost-parity condition. Then, in Subsection 3.2, we show how to solve bounded cost-parity games by solving ω-regular games. The main result of this section is the following theorem. Here, n is the number of vertices, m is the number of edges, and d is the number of colors in the cost-parity game.

▶ **Theorem 6.** *Given an algorithm that solves parity games in time $T(n, m, d)$, there is an algorithm that solves cost-parity games in time $O(n \cdot T(d \cdot n, d \cdot m, d + 2))$.*

3.1 From Cost-Parity Games to Bounded Cost-Parity Games

The cost-parity condition can be rephrased as follows: $\rho \in \mathrm{CostParity}(\Omega)$ if and only if

> there exists a $b \in \mathbb{N}$ such that all but finitely many requests are answered or unanswered with cost less or equal than b and there are only finitely many unanswered requests.

Note that this allows a finite number of unanswered requests with cost ∞. By disallowing this, we obtain a strengthening of the cost-parity condition. Formally, we define the bounded cost-parity condition $\mathrm{BndCostParity}(\Omega)$ to be the set of plays ρ satisfying the following:

> there exists a $b \in \mathbb{N}$ such that every request is answered or unanswered with cost less or equal than b and there are only finitely many unanswered requests.

A game $(\mathcal{A}, \mathrm{BndCostParity}(\Omega))$ is called bounded cost-parity game.

▶ **Example 7.** Consider the game in Figure 1, this time with the bounded cost-parity condition: Player 1 wins from every vertex but g by moving to g and then staying there ad infinitum. Every such play contains a request of color 1 that is unanswered with cost ∞.

Since bounded cost-parity conditions are on the third level of the Borel hierarchy, bounded cost-parity games are determined. Furthermore, the bounded cost-parity condition is indeed a strengthening of the cost-parity condition.

▶ Remark 8. $\mathrm{BndCostParity}(\Omega) \subseteq \mathrm{CostParity}(\Omega)$.

The following lemma implies that being able to solve bounded cost-parity games suffices to solve cost-parity games.

▶ **Lemma 9.** *Let $\mathcal{G} = (\mathcal{A}, \mathrm{CostParity}(\Omega))$ and let $\mathcal{G}' = (\mathcal{A}, \mathrm{BndCostParity}(\Omega))$.*

1. *$W_0(\mathcal{G}') \subseteq W_0(\mathcal{G})$.*
2. *If $W_0(\mathcal{G}') = \emptyset$, then $W_0(\mathcal{G}) = \emptyset$.*

Proof. 1. This follows directly from Remark 8: a winning strategy for Player 0 in $\mathcal{G}'$ from v is also a winning strategy for her in $\mathcal{G}$ from v.

2. Due to determinacy, if $W_0(\mathcal{G}') = \emptyset$, then we have $W_1(\mathcal{G}') = V$, i.e., from every vertex v, Player 1 has a winning strategy τ_v. Consider a play consistent with τ_v starting in v: either, for every b, there is a request that is not answered with cost at most b, or the maximal color seen infinitely often is odd (i.e., there are infinitely many unanswered requests).

We define a strategy τ for Player 1 as follows: it is guided by a vertex v_{cur} and a counter b_{cur}. Assume a play starts in vertex v. Then, we initialize v_{cur} by v and b_{cur} by 1. The strategy plays the strategy $\tau_{v_{\text{cur}}}$ until a request is not answered with cost b_{cur}. If this is the case, then v_{cur} is set to the current vertex, b_{cur} is incremented, and τ plays according to $\tau_{v_{\text{cur}}}$ (forgetting the history of the play constructed so far).

We show that τ is winning (in the cost-parity game) from every vertex, which implies $W_0(\mathcal{G}) = \emptyset$. Let ρ be a play that is consistent with τ and distinguish two cases: if the counter is incremented infinitely often, then ρ is winning for Player 1 in the cost-parity game. On the other hand, if b_{cur} is incremented only finitely often (say to value b), then there is a suffix ρ' of ρ with some first vertex v that is consistent with the strategy τ_v. Since the counter is not incremented during ρ', every request in ρ' is either answered or unanswered with cost at most b. As ρ' is nevertheless winning for Player 1, the maximal color seen infinitely often during ρ' is odd. As ρ' and ρ only differ by a finite prefix, the maximal color seen infinitely often during ρ is the same as in ρ' and therefore odd. Thus, ρ is winning for Player 1 in $\mathcal{G}$. ◀

To conclude this subsection, we show how Lemma 9 can be used to solve cost-parity games, provided we are able to solve bounded cost-parity games (which is the subject of the next subsection). Let $\mathcal{A}$ be an arena and Ω a coloring of its vertices, and let $\mathcal{G} = (\mathcal{A}, \text{CostParity}(\Omega))$. The following algorithm proceeds by iteratively removing parts of $\mathcal{A}$ that are included in the winning region of Player 0 in $\mathcal{G}$. In each step, one computes the winning region of the current arena w.r.t. the bounded cost-parity condition. Due to Lemma 9.1, this is included in the winning region of Player 0 in $\mathcal{G}$. Furthermore, the attractor of this region also belongs to the winning region in $\mathcal{G}$, since the cost-parity condition is prefix-independent. This continues until Player 0's winning region of the current arena w.r.t. the bounded cost-parity condition is empty. In this situation, Player 1 wins everywhere w.r.t. the cost-parity condition in the current arena due to Lemma 9.2. Since Player 0 cannot leave this region (in the original arena $\mathcal{A}$), it is also winning for Player 1 in the original game $\mathcal{G}$.

In Section 4, we prove that Player 0 has uniform positional winning strategies for bounded cost-parity games. Composing such strategies for the regions X_j with positional attractor strategies, we obtain a positional winning strategy for Player 0 in a cost-parity game. In Example 5, we have already seen that Player 1 needs infinite memory to win.

Algorithm 1 A fixed-point algorithm for cost-parity games.

$j \leftarrow 0$; $W_j \leftarrow \emptyset$; $\mathcal{A}_j \leftarrow \mathcal{A}$
repeat
  $j \leftarrow j + 1$
  $X_j \leftarrow W_0(\mathcal{A}_{j-1}, \text{BndCostParity}(\Omega))$
  $W_j \leftarrow W_{j-1} \cup \text{Attr}_0^{\mathcal{A}_{j-1}}(X_j)$
  $\mathcal{A}_j \leftarrow \mathcal{A}_{j-1} \setminus \text{Attr}_0^{\mathcal{A}_{j-1}}(X_j)$
until $X_j = \emptyset$
return W_j

3.2 From Bounded Cost-Parity Games to ω-regular Games

Next, we show how to solve bounded cost-parity games. Let $\mathcal{A}$ be an arena. In this subsection, we assume that no vertex of $\mathcal{A}$ has both incoming increment- and ε-edges. This can be achieved by subdividing every increment-edge $e = (v, v')$: we add a new vertex $\mathrm{sub}(e)$ and replace e by $(v, \mathrm{sub}(e))$ (which is an increment-edge) and by $(\mathrm{sub}(e), v')$ (which is an ε-edge). Now, only the newly added vertices have incoming increment-edges, but they do not have incoming ε-edges. Furthermore, it is easy to see that Player i wins from v in the original game if and only if she wins from v in the modified game (where we color $\mathrm{sub}(e)$ by $\Omega(v')$).

Assuming this convention, we say that a vertex is an increment-vertex, if it has an incoming increment-edge (which implies that all incoming edges have an increment). Let F be the set of increment-vertices and denote the set of plays with finite cost by $\mathrm{coB\ddot{u}chi}(F)$. Furthermore, let $\mathrm{RR}(\Omega)$ denote the set of plays ρ in which every request is answered (no matter at which cost). Using these definitions, let

$$\mathrm{PCRR}(\Omega) = \big(\mathrm{Parity}(\Omega) \cap \mathrm{coB\ddot{u}chi}(F)\big) \cup \mathrm{RR}(\Omega) \ .$$

As $\mathrm{PCRR}(\Omega)$ is ω-regular, a game with winning condition $\mathrm{PCRR}(\Omega)$ is determined and both players have uniform finite-state winning strategies [6]. Furthermore, $\mathrm{PCRR}(\Omega)$ is weaker than $\mathrm{BndCostParity}(\Omega)$.

▶ Remark 10. $V^\omega \setminus \mathrm{PCRR}(\Omega) \subseteq V^\omega \setminus \mathrm{BndCostParity}(\Omega)$.

Note that the converse implication is false, as the request-response condition does not bound the cost between requests and their responses. However, every finite-state winning strategy bounds the distance between requests and their responses, and thereby also the cost.

▶ **Lemma 11.** *Let* $\mathcal{G} = (\mathcal{A}, \mathrm{BndCostParity}(\Omega))$*, and let* $\mathcal{G}' = (\mathcal{A}, \mathrm{PCRR}(\Omega))$*. Then,* $W_i(\mathcal{G}) = W_i(\mathcal{G}')$ *for* $i \in \{0, 1\}$.

Proof. We have $W_1(\mathcal{G}') \subseteq W_1(\mathcal{G})$ due to Remark 10. Thus, it suffices to show $W_0(\mathcal{G}') \subseteq W_0(\mathcal{G})$. So, let σ be a uniform finite-state winning strategy for Player 0 for $\mathcal{G}'$ (which exists, since $\mathrm{PCRR}(\Omega)$ is ω-regular). We argue that σ is also a uniform winning strategy for Player 0 for $\mathcal{G}$: let ρ be consistent with σ, which implies $\rho \in \mathrm{PCRR}(\Omega)$.

If ρ satisfies $\mathrm{Parity}(\Omega)$ and has only finitely many increments, then every request is answered or unanswered with bounded cost and there are only finitely many unanswered requests, i.e., ρ is winning for Player 0 in $\mathcal{G}$. To conclude, we consider the case $\rho \in \mathrm{RR}(\Omega) \setminus (\mathrm{Parity}(\Omega) \cap \mathrm{coB\ddot{u}chi}(F))$. Since σ is a finite-state winning strategy, there is a bound b (which only depends on the size of σ and the size of the arena) such that every request in ρ is answered with cost at most b (if not, then there would also be a play consistent with σ with an unanswered request and with cost ∞. This play would not satisfy $\mathrm{PCRR}(\Omega)$, which yields the desired contradiction). Hence, ρ is winning for Player 0 in $\mathcal{G}$. ◀

So, to determine the winning regions of a bounded cost-parity game it suffices to solve an ω-regular game with winning condition $\mathrm{PCRR}(\Omega)$. This condition can be reduced to a parity condition using a *small* memory structure that keeps track of the largest unanswered request and goes to a special state $\perp$ if this request is answered. Using this memory structure, the $\mathrm{RR}(\Omega)$-condition reduces to a Büchi condition requiring the state $\perp$ to be visited infinitely often. We are now left with the union of a Büchi condition and an intersection of a parity and a co-Büchi condition. First, by coloring the increment-vertices of $\mathcal{A}$ by an odd color that is larger than $\max \Omega(V)$ (and leaving the colors of all other vertices unchanged), we obtain a new parity condition that is equivalent to the intersection of $\mathrm{Parity}(\Omega)$ and $\mathrm{coB\ddot{u}chi}(F)$.

Using a similar construction, we can turn the union of the new parity condition and of the Büchi condition into an equivalent parity condition. Thus, to determine the winning regions of a game with winning condition $\mathrm{PCRR}(\Omega)$, if suffices to solve a linearly larger parity game. This also concludes the proof of Theorem 6: Algorithm 1 terminates after at most n iterations and solves a *small* parity game in each iteration.

Furthermore, from the previous observations we obtain an upper bound on the memory requirements for both players, which we improve in the next section.

▶ Remark 12. In every bounded cost-parity game, both players have uniform finite-state winning strategies with $\ell + 1$ memory states, where ℓ is the number of odd colors in $\Omega(V)$.

The winning regions of parity games can be determined in non-deterministic polynomial time by guessing both regions $W_i(\mathcal{G})$ and positional strategies σ_i for both players and then verifying in (deterministic) polynomial time whether σ_i is a uniform winning strategy for Player i from $W_i(\mathcal{G})$. Algorithm 1 solves cost-parity games by solving at most n parity games, which have at most $(\ell+1)\cdot n$ vertices, $(\ell+1)\cdot m$ edges, and $d+2$ colors, where n, m, d, and ℓ denote the number of vertices, edges, colors, and odd colors in the cost-parity game. Thus, the algorithm runs in non-deterministic polynomial time. Together with a dual argument this implies the following result.

▶ **Theorem 13.** *The following problems are in* $\mathbf{NP} \cap \mathbf{coNP}$*:*

1. *Given a bounded cost-parity game $\mathcal{G}$ and a vertex v, is $v \in W_0(\mathcal{G})$?*
2. *Given a cost-parity game $\mathcal{G}$ and a vertex v, is $v \in W_0(\mathcal{G})$?*

4 Half-Positional Determinacy of (Bounded) Cost-Parity Games

Next, we show that Player 0 has positional winning strategies in (bounded) cost-parity games.

▶ **Theorem 14.** *In cost-parity games, Player 0 has uniform positional winning strategies.*

As we have already explained while proving the correctness of Algorithm 1, the existence of positional winning strategies for bounded cost-parity games implies Theorem 14. Thus, the remainder of this section is devoted to proving the following lemma.

▶ **Lemma 15.** *In bounded cost-parity games, Player 0 has uniform positional winning strategies.*

For bounded cost-parity games we have already proved the existence of uniform finite-state winning strategies (see Remark 12). Hence, it remains to show how to eliminate the memory. To this end, we define a so-called scoring function for bounded cost-parity games that measures the quality of a play prefix (from Player 0's vantage point) by keeping track of the largest unanswered request, the number of increment-edges traversed since it was raised, and how often each odd color was seen since the last increment-edge.

In the following, fix an arena $(V, V_0, V_1, E, \mathrm{Cst})$ and a coloring function $\Omega\colon V \to \mathbb{N}$. Let $\Omega(V) \subseteq \{0,\dots,d\}$, where we assume d to be odd, and define $\ell = \frac{d+1}{2}$ to denote the number of odd colors in $\{0,\dots,d\}$. The (score-) sheet of a play prefix w, denoted by $\mathrm{Sh}(w)$, is a vector $(c, n, s_d, s_{d-2}, \dots, s_1) \in \mathbb{N}^{2+\ell}$ containing the following information:

- c denotes the largest unanswered request in w, i.e., the largest odd color in w that is not followed by a color in $\mathrm{Ans}(c)$.
- n denotes the cost of the suffix starting with the *first* unanswered occurrence of c in w.
- $s_{c'}$ (here, c' is odd) denotes the number of times c' occurred in w since the first unanswered occurrence of c, since the last increment-edge was traversed, or since the last time a color larger than c' occurred in w, depending on which happened last.

Finally, we use the empty sheet $\bot$ for play prefixes without unanswered requests.

The reversed ordering of the score values $s_d, \dots, s_1$ in the sheets is due to the max-parity condition, in which larger colors are more important than smaller ones. This is reflected by the fact that we compare sheets in the lexicographical order induced by $<$ on its components and add $\bot$ as minimal element. For example, $(3,3,0,1,1) < (3,3,1,0,7)$ and $\bot < s$ for every sheet $s \neq \bot$. As usual, we write $s \leq s'$, if $s = s'$ or $s < s'$. We say that a sheet $(c, n, s_d, \dots, s_1)$ is bounded by $b \in \mathbb{N}$, if we have $n \leq b$ and $s_c \leq b$ for every c. Also, $\bot$ is bounded by every b. The following lemma shows that Sh is a congruence w.r.t. $\leq$. Here $\mathrm{Lst}(x)$ denotes the last vertex of the non-empty finite play x.

▶ **Lemma 16.** *If* $\mathrm{Lst}(x) = \mathrm{Lst}(y)$ *and* $\mathrm{Sh}(x) \leq \mathrm{Sh}(y)$*, then* $\mathrm{Sh}(xv) \leq \mathrm{Sh}(yv)$ *for every* $v \in V$*.*

We begin the proof of Lemma 15 by noting that the sheets of a play ρ being bounded is a sufficient condition for ρ satisfying the bounded cost-parity condition.

▶ **Lemma 17.** *If there exists a bound* b *such that the sheets of all prefixes of a play* ρ *are bounded by* b*, then* $\rho \in \mathrm{BndCostParity}(\Omega)$*.*

The next lemma shows that finite-state winning strategies uniformly bound the sheets.

▶ **Lemma 18.** *Let* σ *be a uniform finite-state winning strategy* σ *(of size* m*) for Player* 0 *in a bounded cost-parity game* $\mathcal{G} = (\mathcal{A}, \mathrm{BndCostParity}(\Omega))$*. Furthermore, let* ρ *be starting in* $W_0(\mathcal{G})$ *and be consistent with* σ*. Then, the sheets of all prefixes of* ρ *are bounded by* $m \cdot |V|$*.*

Now we are able to prove our main technical result of this section: Player 0 has a uniform positional winning strategy in every bounded cost-parity game (and therefore also in every cost-parity game).

Proof of Lemma 15. Fix some uniform finite-state winning strategy σ' for Player 0 in a bounded cost-parity game $\mathcal{G}$. For every $v \in W_0(\mathcal{G})$, let P_v denote the set of play prefixes that begin in $W_0(\mathcal{G})$, are consistent with σ', and end in v. Due to Lemma 18, the sheets of the prefixes in P_v are bounded by some b. Thus, for every v the set $\{\mathrm{Sh}(w) \mid w \in P_v\}$ is finite. Hence, there exists a play prefix $\max_v \in P_v$ such that $\mathrm{Sh}(w) \leq \mathrm{Sh}(\max_v)$ for every $w \in P_v$.

We define a uniform positional strategy σ by $\sigma(wv) = \sigma'(\max_v)$ and claim that it is a uniform winning strategy for $\mathcal{G}$. An inductive application of Lemma 16 shows that we have $\mathrm{Sh}(\rho_0 \cdots \rho_n) \leq \mathrm{Sh}(\max_{\rho_n})$ for every n and every play ρ that is consistent with σ. Hence, the sheets of ρ are bounded by b, which implies $\rho \in \mathrm{BndCostParity}(\Omega)$ due to Lemma 17. ◀

5 Cost-Streett Games

In this section, we introduce cost-Streett games which generalize both Streett games and finitary Streett games [8]. We first present an algorithm to solve these games following the same ideas as in the previous sections, and prove **EXPTIME**-completeness of the corresponding decision problem. From our algorithm, we deduce finite-state determinacy for Player 0, while Player 1 needs infinite memory in general.

To simplify our notation, let $[d] = \{0, \dots, d-1\}$. A d-dimensional cost-arena has the form $\mathcal{A} = (V, V_0, V_1, E, (\mathrm{Cst}_\ell)_{\ell \in [d]})$ where the first four components are as usual and where each Cst_ℓ is a mapping from E to $\{\varepsilon, i\}$. Again, each of these functions induces a cost on (infixes of) plays, denoted by Cst_ℓ as well. Let $\Gamma = (Q_\ell, P_\ell)_{\ell \in [d]}$ be a collection of (Streett) pairs of subsets of V, i.e., $Q_\ell, P_\ell \subseteq V$. We define

$$\mathrm{StCor}_\ell(\rho, k) = \begin{cases} 0 & \text{if } \rho_k \notin Q_\ell, \\ \inf\{\mathrm{Cst}_\ell(\rho_k \cdots \rho_{k'}) \mid \rho_{k'} \geq \rho_k \text{ and } \rho_{k'} \in P_\ell\} & \text{if } \rho_k \in Q_\ell, \end{cases}$$

where we use $\inf \emptyset = \infty$, and $\mathrm{StCor}(\rho, k) = \max\{\mathrm{StCor}_\ell(\rho, k) \mid \ell \in [d]\}$. Using this, we define the following three winning conditions:

- the (classical) Streett condition $\mathrm{Streett}(\Gamma)$ requires for every ℓ that P_ℓ is visited infinitely often if Q_ℓ is visited infinitely often.
- the request-response condition $\mathrm{RR}(\Gamma)$ requires for every ℓ that every visit to Q_ℓ is answered by a later visit to P_ℓ.
- the cost-Streett condition $\mathrm{CostStreett}(\Gamma) = \{\rho \mid \limsup_{k\to\infty} \mathrm{StCor}(\rho, k) < \infty\}$.

A cost-Streett game $(\mathcal{A}, \Gamma)$ consists of a d-dimensional arena and a collection Γ of d Streett pairs. Streett and request-response games are defined accordingly. As for cost-parity games, a cost-Streett game in which every edge is an increment-edge (w.r.t. all Cst_ℓ) is a finitary Streett game and a cost-Streett game in which every edge is an epsilon-edge (w.r.t. all Cst_ℓ) is a Streett game. Furthermore, just as classical Streett games subsume parity games, cost-Streett games subsume cost-parity games.

▶ Remark 19. Cost-Streett games are determined.

Our main theorem is proved along the same lines as Theorem 6.

▶ **Theorem 20.** *To solve a cost-Streett game with n vertices, it suffices to solve n Streett games which are exponentially larger (but only in the number of Streett pairs).*

We say that the requests at position k are answered with cost c, if $\mathrm{StCor}(\rho, k) = c$; that the requests are unanswered with cost c, if $\mathrm{StCor}(\rho, k) = \infty$, but there are at most c many increment-edges after position k (w.r.t. all Cst_ℓ such that $\rho_k \in Q_\ell$); and that the requests are unanswered with cost ∞, if $\mathrm{StCor}(\rho, k) = \infty$ and there are infinitely many increment-edges after position k (w.r.t. some Cst_ℓ such that $\rho_k \in Q_\ell$).

As in the case of cost-parity games, we begin by introducing a strengthening of the cost-Streett condition. The bounded cost-Streett condition, denoted by $\mathrm{BndCostStreett}(\Gamma)$, is the set of plays ρ that satisfy the following condition:

> there exists a b such that all requests are answered or unanswered with cost at most b, and there are only finitely many unanswered requests.

▶ **Lemma 21.** *Let $\mathcal{G} = (\mathcal{A}, \mathrm{CostStreett}(\Gamma))$, and let $\mathcal{G}' = (\mathcal{A}, \mathrm{BndCostStreett}(\Gamma))$.*

1. *$W_0(\mathcal{G}') \subseteq W_0(\mathcal{G})$.*
2. *If $W_0(\mathcal{G}') = \emptyset$, then $W_0(\mathcal{G}) = \emptyset$.*

The proof is analogously to the one for Lemma 9. Also, Algorithm 1 (where X_j is now Player 0's winning region in the bounded cost-Streett game) works for this pair of winning conditions as well. To solve bounded cost-Streett games, we again assume for every ℓ that no vertex has both an incoming increment-edge (w.r.t. Cst_ℓ) and an incoming epsilon-edge (again, w.r.t. Cst_ℓ). Assuming this, let F_ℓ denote the vertices with incoming increment-edges w.r.t. Cst_ℓ. Then, $\mathrm{coBüchi}(F_\ell) = \{\rho \mid \mathrm{Cst}_\ell(\rho) < \infty\}$ is the set of plays with finitely many increment-edges w.r.t. Cst_ℓ. Finally, we define the ω-regular condition

$$\mathrm{SCRR}(\Gamma) = \bigcap\nolimits_{\ell\in[d]} \left[\left(\mathrm{Streett}(Q_\ell, P_\ell) \cap \mathrm{coBüchi}(F_\ell)\right) \cup \mathrm{RR}(Q_\ell, P_\ell) \right] .$$

▶ **Lemma 22.** *Let $\mathcal{G} = (\mathcal{A}, \mathrm{BndCostStreett}(\Gamma))$, and let $\mathcal{G}' = (\mathcal{A}, \mathrm{SCRR}(\Gamma))$. Then, $W_i(\mathcal{G}) = W_i(\mathcal{G}')$ for $i \in \{0, 1\}$.*

The proof is again analogously to the one for Lemma 11 and relies on finite-state determinacy of ω-regular games. To solve $(\mathcal{A}, \mathrm{SCRR}(\Gamma))$ one can reduce this game to a

classical Streett game with $2d$ Streett pairs using a memory structure of size 2^d to keep track of open requests. Hence, using the algorithm for Streett games from [19], we can solve the resulting game in exponential time (in the size of $(\mathcal{A}, \mathrm{SCRR}(\Gamma))$), even though the arena is of exponential size (but only in d). Together with the **EXPTIME**-hardness of solving finitary Streett games[1], which are a special case, we obtain the following result.

▶ **Theorem 23.** *The problem "Given a cost-Streett game $\mathcal{G}$ and a vertex v, is $v \in W_0(\mathcal{G})$?" is* **EXPTIME***-complete.*

Furthermore, the reduction described above (and the memory requirements for Streett games [12]) yields upper bounds on the memory requirements in (bounded) cost-Streett. Player 1 needs in general infinite memory in cost-Streett games.

▶ **Corollary 24.**

- *Player 0 has finite-state winning strategies of size $2^d \cdot (2d)!$ in bounded cost-Streett games and cost-Streett games.*
- *Player 1 has finite-state winning strategies of size 2^d in bounded cost-Streett games.*

6 Conclusion

We introduced infinite games with cost conditions, generalizing both classical conditions and finitary conditions. For cost-parity games, we proved half-positional determinacy and that solving these games is not harder than solving parity games. Furthermore, the corresponding decision problem is in $\mathbf{NP} \cap \mathbf{coNP}$. For cost-Streett games, we showed that Player 0 has finite-state winning strategies and that solving these games is not harder than solving finitary Streett games and can be done by solving linearly many (classical) Streett games of exponential size (in the number of Streett pairs). Our results unify the previous results on both classical and finitary variants. Table 1 sums up all these results.

winning condition	computational complexity	memory Player 0	memory Player 1
parity	$\mathbf{NP} \cap \mathbf{coNP}$	positional	positional
finitary parity	**PTIME**	positional	infinite
cost-parity	$\mathbf{NP} \cap \mathbf{coNP}$	positional	infinite
Streett	**coNP**-complete	finite	positional
finitary Streett	**EXPTIME**-complete	finite	infinite
cost-Streett	**EXPTIME**-complete	finite	infinite

Table 1 Overview of results

There are at least three directions to extend these results: first, our winning conditions do not cover all acceptance conditions (for automata) discussed in [3, 20]. In ongoing research, we investigate whether our techniques are applicable to these more expressive conditions and to winning conditions specified in weak-MSO with the unbounding quantifier [2, 4]. Second, one could consider infinite arenas, e.g., configuration graphs of pushdown systems. This is already open for finitary games and requires new ideas, since our techniques rely heavily on the finiteness of the arena. Finally, one could add decrement-edges.

[1] Shown in unpublished work by Chatterjee, Henzinger, and Horn, obtained by slightly modifying the proof of **EXPTIME**-hardness of solving request-response games [9].

References

1. Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *CSL*, volume 3210 of *LNCS*, pages 41–55. Springer, 2004.
2. Mikołaj Bojańczyk. Weak MSO with the unbounding quantifier. *Theory Comput. Syst.*, 48(3):554–576, 2011.
3. Mikołaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In LICS [16], pages 285–296.
4. Mikołaj Bojańczyk and Szymon Toruńczyk. Weak MSO+U over infinite trees. In Christoph Dürr and Thomas Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 648–660. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
5. Tomás Brázdil, Krishnendu Chatterjee, Antonín Kucera, and Petr Novotný. Efficient controller synthesis for consumption games with multiple resource types. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, volume 7358 of *LNCS*, pages 23–38. Springer, 2012.
6. J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:pp. 295–311, 1969.
7. Krishnendu Chatterjee and Laurent Doyen. Energy parity games. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (2)*, volume 6199 of *LNCS*, pages 599–610. Springer, 2010.
8. Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. Finitary winning in ω-regular games. *ACM Trans. Comput. Log.*, 11(1), 2009.
9. Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. The complexity of request-response games. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *LNCS*, pages 227–237. Springer, 2011.
10. Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (2)*, volume 5556 of *LNCS*, pages 139–150. Springer, 2009.
11. Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79. IEEE Computer Society, 2010.
12. Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110, 1997.
13. E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, 1999.
14. Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jirí Srba. Energy games in multiweighted automata. In Antonio Cerone and Pekka Pihlajasaari, editors, *ICTAC*, volume 6916 of *LNCS*, pages 95–115. Springer, 2011.
15. Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 24(2):233–244, 1982.
16. *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*. IEEE Computer Society, 2006.
17. Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.
18. David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.
19. Nir Piterman and Amir Pnueli. Faster solutions of Rabin and Streett games. In LICS [16], pages 275–284.
20. Michael Vanden Boom. Weak cost monadic logic over infinite trees. In Filip Murlak and Piotr Sankowski, editors, *MFCS*, volume 6907 of *LNCS*, pages 580–591. Springer, 2011.

Super-Fast 3-Ruling Sets*

Kishore Kothapalli[1] and Sriram Pemmaraju[2]

1 **International Institute of Information Technology, Hyderabad, India 500 032** kkishore@iiit.ac.in

2 **Department of Computer Science, The University of Iowa, Iowa City, IA 52242-1419, USA,** sriram-pemmaraju@uiowa.edu

Abstract

A t-ruling set of a graph $G = (V, E)$ is a vertex-subset $S \subseteq V$ that is independent and satisfies the property that every vertex $v \in V$ is at a distance of at most t from some vertex in S. A *maximal independent set (MIS)* is a 1-ruling set. The problem of computing an MIS on a network is a fundamental problem in distributed algorithms and the fastest algorithm for this problem is the $O(\log n)$-round algorithm due to Luby (SICOMP 1986) and Alon et al. (J. Algorithms 1986) from more than 25 years ago. Since then the problem has resisted all efforts to yield to a sub-logarithmic round algorithm. There has been recent progress on this problem, most importantly an $O(\log \Delta \cdot \sqrt{\log n})$-round algorithm on graphs with n vertices and maximum degree Δ, due to Barenboim et al. (to appear FOCS 2012). The time complexity of this algorithm is sub-logarithmic for $\Delta = 2^{o(\sqrt{\log n})}$.

We approach the MIS problem from a different angle and ask if $O(1)$-ruling sets can be computed faster than the currently known fastest algorithm for an MIS? As an answer to this question, we show how to compute a 2-ruling set of an n-vertex graph in $O((\log n)^{3/4})$ rounds. We also show that the above result can be improved for special classes of graphs. For instance, on high girth graphs (girth 6 or more), trees, and graphs of bounded arboricity, we show how to compute 3-ruling sets in $\exp(O(\sqrt{\log \log n}))$ rounds, $O((\log \log n)^2 \cdot \log \log \log n)$ rounds, and $O((\log \log n)^3)$ rounds, respectively.

Our main technique involves randomized sparsification that rapidly reduces the graph degree while ensuring that every deleted vertex is close to some vertex that remains. This technique may have further applications in other contexts, e.g., in designing sub-logarithmic distributed approximation algorithms. Our results raise intriguing questions about how quickly an MIS (or 1-ruling sets) can be computed, given that 2-ruling sets can be computed in sub-logarithmic rounds.

1998 ACM Subject Classification G.2.2 Graph algorithms

Keywords and phrases MIS, ruling sets, graph sparsification, distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.136

1 Introduction

Symmetry breaking is a fundamental theme in distributed computing and a classic example of symmetry breaking arises in the computation of a *maximal independent set* (MIS) of a given graph. About 25 years ago Alon et al. [1] and Luby [12] independently devised randomized algorithms for the MIS problem, running in $O(\log n)$ communication rounds. Since then, all

* Part of this work was done while the first author was visiting the University of Iowa as an Indo-US Science and Technology Forum Research Fellow. The work of the second author is supported in part by National Science Foundation grant CCF 0915543.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 136–147
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

attempts to devise an algorithm for MIS that runs in *sub-logarithmic* rounds (for general graphs) have failed. Recently, Kuhn et al. [10] proved that there exist n-vertex graphs for which any distributed algorithm, even randomized, that solves the MIS problem requires $\Omega(\sqrt{\log n})$ communication rounds. Closing this gap between the $O(\log n)$ upper bound and the $\Omega(\sqrt{\log n})$ lower bound is one of the fundamental challenges in distributed computing.

There has been some exciting recent progress in closing this gap. Barenboim et al. [5] present an algorithm that runs in $O(\log \Delta \sqrt{\log n})$ rounds on n-vertex graphs with maximum degree Δ. This is sub-logarithmic for $\Delta \in 2^{o(\sqrt{\log n})}$. This result uses techniques developed in a paper by Kothapalli et al. [8] for deriving an $O(\sqrt{\log n})$-round algorithm for computing an $O(\Delta)$-coloring of a n-vertex graph with maximum degree Δ. Barenboim et al. [5] also present an algorithm for computing an MIS on trees in $O(\sqrt{\log n \log\log n})$ rounds. This is a small improvement over an algorithm from PODC 2011 for computing an MIS on trees due to Lenzen and Wattenhofer [11] that runs in $O(\sqrt{\log n} \cdot \log\log n)$ rounds. Barenboim et al. extend their result on MIS on trees to graphs with girth at least 6 and to graphs with bounded arboricity.

A problem closely related to MIS, that also involves symmetry breaking at its core, is the problem of computing t-ruling sets. A *t-ruling set* of a graph $G = (V, E)$ is an independent subset S of vertices with the property that every vertex $v \in V$ is at a distance of at most t from some vertex in S. Thus an MIS is a 1-ruling set[1]. In this paper we investigate the distributed complexity of the problem of computing t-ruling sets for $t = O(1)$ with the aim of determining whether an $O(1)$-ruling set can be computed more efficiently than an MIS. For general graphs and for various graph subclasses we show that it is indeed possible to compute t-ruling sets, for small constant t, in time that is much smaller than the best currently known running time for a corresponding MIS algorithm. In our first result, we present an algorithm that computes a 2-ruling set in $O((\log n)^{3/4})$ rounds on general graphs. Thus we have a sub-logarithmic algorithm for a seemingly minor "relaxation" of the MIS problem. We improve on this result substantially for trees, graphs of girth at least 6, and graphs of bounded arboricity. For all these subclasses, we present algorithms for computing 3-ruling sets whose runtime (in rounds) is exponentially faster than the fastest currently known corresponding MIS algorithms. For example, for trees our algorithm computes a 3-ruling set in $O((\log\log n)^2 \cdot \log\log\log n)$ communication rounds, whereas the fastest currently known algorithm for MIS on trees takes $O(\sqrt{\log n \log\log n})$ rounds [5].

Our work raises intriguing questions on the possibility of faster MIS algorithms and on the separation between the distributed complexity of $O(1)$-ruling sets and MIS. For example, could we design algorithms for MIS that first compute a 2- or 3-ruling set and then quickly convert that subset to a 1-ruling set? Is it possible that there are MIS algorithms for trees and related graph subclasses that run in $O(\text{poly}(\log\log n))$ rounds? Alternately, could the MIS problem be *strictly* harder than the problem of computing a t-ruling set for some small constant t?

Our results should also be viewed in the context of results by Gfeller and Vicari [7]. These authors showed how to compute in $O(\log\log n)$ rounds a vertex-subset T of a given n-vertex graph $G = (V, E)$ such that (i) every vertex is at most $O(\log\log n)$ hops from some vertex in T and (ii) the subgraph induced by T has maximum degree $O(\log^5 n)$. One can use the Barenboim et al. $O(\log \Delta \sqrt{\log n})$-round MIS algorithm on $G[T]$ and sparsify T into an $O(\log\log n)$-ruling set in an additional $O(\sqrt{\log n} \cdot \log\log n)$ rounds. Thus, by combining the

[1] In the definition of Gfeller and Vicari [7], a t-ruling set need not be independent, and what we call a t-ruling set, they call an *independent* t-ruling set.

Gfeller-Vicari algorithm with the Barenboim et al. algorithm one can compute an $O(\log\log n)$-ruling set in general graphs in $O(\sqrt{\log n}\cdot\log\log n)$ rounds. Our result can be viewed as extending the Gfeller-Vicari result by using $t = O(1)$ instead of $t = O(\log\log n)$. Also worth noting is the fact that Gfeller and Vicari use their $O(\log\log n)$-ruling set computation as an intermediate step to computing an MIS on *growth-bounded graphs*. While the techniques that work for growth-bounded graphs do not work for general graphs or for the other graph subclasses we consider, this suggests the possibility of getting to an MIS via a t-ruling set for small t.

Our technique involves a rapid sparsification of the graph while ensuring that nodes that are removed from further consideration are close (within one or two hops) to some remaining node. Using this technique we show how to reduce the degrees of graphs rapidly and after sufficiently reducing the degrees, we can apply MIS algorithms due to Barenboim et al. [5] that take advantage of the low maximum degree. For example, given a graph $G = (V, E)$ and a parameter ϵ, $0 < \epsilon < 1$, our sparsification procedure can run in $O\left(\frac{\log\Delta}{(\log n)^{\epsilon}}\right)$ rounds and partition V into subsets M and W such that with high probability (i) $G[M]$ has maximum degree $O(2^{(\log n)^{\epsilon}})$ and (ii) every vertex in W has a neighbor in M. At this stage, we can apply the MIS algorithm of Barenboim et al. [5] that runs in $O(\log\Delta\cdot\sqrt{\log n})$ rounds on $G[M]$. Since $\Delta(G[M]) = O(2^{(\log n)^{\epsilon}})$, this step takes $O((\log n)^{1/2+\epsilon})$ rounds, leading to a 2-ruling set algorithm that runs in $O\left(\frac{\log\Delta}{(\log n)^{\epsilon}} + (\log n)^{1/2+\epsilon}\right)$ rounds. Picking $\epsilon = 1/4$ yields the $O((\log n)^{3/4})$ rounds 2-ruling set algorithm mentioned above. We use a similar rapid sparsification approach to derive faster ruling set algorithms for different graph subclasses. We believe that the sparsification technique may be of independent interest in itself, especially in designing distributed approximation algorithms that run in sub-logarithmic number of rounds.

1.1 Model

We consider distributed systems that can be modeled by a graph $G = (V, E)$ with the vertices representing the computational entities and the edges representing communication links between pairs of computational entities. We use the standard synchronous, message passing model of communication in which each node, in each round, can send a possibly distinct message along each incident edge. All of our algorithms are structured as a series of "sparsification" steps interleaved with calls to subroutines implementing MIS algorithms on low degree graphs, due to Barenboim et al. [5]. During the sparsification steps, each node only needs to inform its neighbors of its membership in some set and therefore each node only needs to send the same single bit to all of its neighbors. Therefore, communication during the sparsification steps can be viewed as occuring in in a fairly restrictive communication model in which each node is only allowed to (locally) broadcast a single bit to all neighbors. However, some of the MIS algorithms in Barenboim et al. [5] run in the $\mathcal{LOCAL}$ model, which allows each node to send a message of arbitrary size to each neighbor in each round. Thus, due to their dependency on the MIS algorithms of Barenboim et al. [5], the algorithms in this paper also require the use of the $\mathcal{LOCAL}$ model.

1.2 Definitions and Notation

Given a graph $G = (V, E)$, we denote by $N(v)$ the neighborhood of v and by $\deg_G(v)$ the quantity $|N(v)|$. Let $\text{dist}_G(u, v)$ refer to the shortest distance between any two vertices u and v in G. For a subset of vertices $V' \subseteq V$, let $G[V']$ be the subgraph induced by the subset V'.

Our calculations make use of Chernoff bounds for tail inequalities on the sum of independent random variables. In particular, let $X := \sum_{i=1}^{n} X_i$ with $E[X_i] = p$ for each $1 \leq i \leq n$. The upper tail version of Chernoff bounds that we utilize is: $\Pr[X \geq E[X] \cdot (1+\epsilon)] \leq \exp(-E[X]\epsilon^2/3)$ for any $0 < \epsilon < 1$.

In our work, we derive a 3-ruling set algorithm for graphs with bounded arboricity. Let the *density* of a graph $G = (V, E)$, $|V| \geq 2$, be the ratio $\lceil |E|/(|V|-1) \rceil$. Let the density of a single-vertex graph be 1. The *arboricity* of a graph $G = (V, E)$, denoted $a(G)$, can be defined as $a(G) := \max\{density(G') \mid G' \text{ is a subgraph of } G\}$. By the celebrated Nash-Williams decomposition theorem [14], the arboricity of a graph is exactly equal to the minimum number of forests that its edge set can be decomposed into. For example, trees have arboricity one. The family of graphs with arboricity $a(G) = O(1)$ includes all planar graphs, graphs with treewidth bounded by a constant, graphs with genus bounded by a constant, and the family of graphs that exclude a fixed minor. A property of graphs with arboricity $a(G)$ that has been found useful in distributed computing [2, 3, 4] is that the edges of such graphs can be oriented so that each node has at most $a(G)$ incident edges oriented away from it. However, finding such an orientation takes $\Omega\left(\frac{\log n}{\log a(G)}\right)$ rounds by a lower bound result due to Barenboim and Elkin [2] and since we are interested in sub-logarithmic algorithms, we cannot rely on the availability of such an orientation for $a(G) = O(1)$.

1.3 Our Results

Here we summarize the results in this paper.

1. An algorithm, that with high probability, computes a 2-ruling set on general graphs in $O\left(\frac{\log \Delta}{(\log n)^{\epsilon}} + (\log n)^{1/2+\epsilon}\right)$ rounds for any $0 < \epsilon < 1$. Substituting $\epsilon = 1/4$ into this running time expression simplifies it to $O((\log n)^{3/4})$.
2. An algorithm, that with high probability, computes a 3-ruling set on graphs of girth at least 6 in $\exp(O(\sqrt{\log \log n}))$ rounds.
3. An algorithm, that with high probability, computes a 3-ruling set in $O((\log \log n)^2 \log \log \log n)$ rounds on trees.
4. An algorithm, that with high probability, computes a 3-ruling set on graphs of bounded arboricity in $O((\log \log n)^3)$ rounds.

Note that all our results run significantly faster than currently known fastest corresponding algorithms for MIS. In fact, for trees and graphs of bounded arboricity, our results improve the corresponding results exponentially. This is illustrated further in Table 1.

Table 1 Comparison of the best known runtimes of distributed algorithms for MIS, $O(\log \log n)$-ruling sets, and 3-ruling sets. It should be noted that the algorithm for general graphs described in this paper computes a 2-ruling set. Also, we use the notation $\tilde{O}(f(n))$ as a short form for $O(f(n) \cdot \text{polylog}(f(n)))$.

Graph Class	**MIS [5]**	$O(\log \log n)$**-ruling sets [7]**	**3-ruling set** [This Paper]
General	$O(\log \Delta \cdot \sqrt{\log n})$	$O(\sqrt{\log n} \cdot \log \log n)$	$O((\log n)^{3/4})$
Trees	$\tilde{O}(\sqrt{\log n})$		$\tilde{O}((\log \log n)^2)$
Girth ≥ 6	$O(\log \Delta \log \log n + e^{O(\sqrt{\log \log n})})$		$e^{O(\sqrt{\log \log n})}$
Bounded arboricity ($a = O(1)$)	$O(\log \Delta(\log \Delta + \frac{\log \log n}{\log \log \log n}))$		$O((\log \log n)^3)$

1.4 Related Work

The work most closely related to ours, which includes the recent work of Barenboim et al. [5] and the work of Gfeller and Vicari [7], has already been reviewed earlier in this section.

Other work on the MIS problem that is worth mentioning is the elegant MIS algorithm of Métivier et al. [13]. In this algorithm, each vertex picks a real uniformly at random from the interval $[0, 1]$ and joins the MIS if its chosen value is a local maxima. This can be viewed as a variant of Luby's algorithm [12] and like Luby's algorithm, runs in $O(\log n)$ rounds. Due to its simplicity, this MIS algorithm is used in part by the MIS algorithm on trees by Lenzen and Wattenhofer [11] and also by Barenboim et al. [5].

The MIS problem on the class of growth-bounded graphs has attracted fair bit of attention [9, 7, 15]. Growth-bounded graphs have the property that the r-neighborhood of any vertex v has at most $O(r^c)$ independent vertices in it, for some constant $c > 0$. In other words, the rate of the growth of independent sets is polynomial in the radius of the "ball" around a vertex. Schneider and Wattenhofer [15] showed that there is a deterministic MIS algorithm on growth-bounded graphs that runs in $O(\log^* n)$ rounds. Growth-bounded graphs have been used to model wireless networks because the number of independent vertices in any spatial region is usually bounded by the area or volume of that region. In contrast to growth-bounded graphs, the graph subclasses we consider in this paper tend to have arbitrarily many independent vertices in any neighborhood.

Fast algorithms for $O(1)$-ruling sets may have applications in distributed approximation algorithms. For example, in a recent paper by Berns et al. [6] a 2-ruling set is computed as a way of obtaining a $O(1)$-factor approximation to the metric facility location problem. Our work raises questions about the existence of sub-logarithmic round algorithms for problems such as minimum dominating set, vertex cover, etc., at least for special graph classes.

1.5 Organization of the Paper

The rest of the paper is organized as follows. Section 2 shows our result for general graphs. Section 3 shows our results for graphs of girth at least 6, and for trees. Section 4 extends the results of Section 3 to graphs of arboricity bounded by a constant.

2 2-Ruling Sets in General Graphs

In this section we describe Algorithm RulingSet-GG, that runs in sub-logarithmic rounds and computes a 2-ruling set in general graphs. The reader is encouraged to consult the pseudocode of this algorithm while reading the following text. Let f be the quantity $2^{(\log n)^\epsilon}$ for some parameter $0 < \epsilon < 1$. Let i^* be the smallest positive integer such that $f^{i^*+1} \geq \Delta$. Thus $i^* = \lceil \log_f \Delta \rceil - 1$. It is also useful to note that $i^* = O\left(\frac{\log \Delta}{(\log n)^\epsilon}\right)$. The algorithm proceeds in *stages* and there are i^* stages, indexed by $i = 1, 2, \ldots, i^*$. In Stage i, all "high degree" vertices, i.e., vertices with degrees greater than $\frac{\Delta}{f^i}$, are processed. Roughly speaking, in each stage we peel off from the "high degree" vertex set, a subgraph with degree bounded above by $O(f \cdot \log n)$. Following this we also peel off all neighbors of this subgraph. More precisely, in Stage i each "high degree" vertex joins a set M_i with probability $\frac{6 \log n \cdot f^i}{\Delta}$ (Line 6). Later we will show (in Lemma 1) that with high probability any vertex that is in V at the start of Stage i has degree at most Δ / f^{i-1}. (This is trivially true for $i = 1$.) Therefore, it is easy to see that any vertex in the graph induced by M_i has expected degree at most $O(f \cdot \log n)$. In fact, this is true with high probability, as shown in Lemma 2. This degree bound allows the efficient computation of an MIS on the subgraph induced by M_i. Following

the identification of the set M_i, all neighbors of M_i that are outside M_i are placed in a set W_i (Line 9). Both sets M_i and W_i are then deleted from the vertex set V. The sets W_i play a critical role in our algorithm. For one, given the probability $\frac{6\log n \cdot f^i}{\Delta}$ of joining M_i, we can show that with high probability every "high degree" vertex ends up either in M_i or in W_i. This ensures that all "high degree" vertices are deleted from V in each Stage i. Also, the sets W_i act as "buffers" between the M_i's ensuring that there are no edges between M_i and $M_{i'}$ for $i \neq i'$. As a result the graph induced by $\cup_i M_i$ also has low degree, i.e., $O(f \cdot \log n)$. Therefore, we can compute an MIS on the graph induced by $\cup_i M_i$ in "one shot" rather than deal with each of the graphs induced by $M_1, M_2, \ldots$ one by one.

Given the way in which "high degree" vertices disappear from V, at the end of all i^* stages, the graph G induced by vertices that still remain in V would have shrunk to the point where the maximum degree of a vertex in G is $O(f)$. The algorithm ends by computing an MIS on the graph induced by $V \cup (\cup_i M_i)$. As mentioned before, the M_i's do not interact with each other or with V and therefore the degree of the graph induced by $(\cup_i M_i) \cup V$ is $O(f \cdot \log n)$. We use the MIS algorithm due of Barenboim et al. [5] that runs in $O(\log \Delta \cdot \sqrt{\log n})$ rounds for this purpose. Since $\Delta = O(f \cdot \log n)$ and $f = 2^{(\log n)^\epsilon}$, this step runs in $O((\log n)^{\frac{1}{2}+\epsilon})$ rounds. In the algorithm described below, we denote by `MIS-LOWDEG` the subroutine that implements the Barenboim et al. algorithm. We use H to denote a static copy of the input graph G.

Algorithm RulingSet-GG($G = (V, E)$)

1. $f \leftarrow 2^{(\log n)^\epsilon}$; $H \leftarrow G$
2. **for** $i \leftarrow 1, 2, \ldots, i^*$ **do**
   /* Stage i ***/**
3. $M_i \leftarrow \emptyset$; $W_i \leftarrow \emptyset$;
4. **for** each $v \in V$ *in parallel* **do**
5. **if** $\deg_G(v) > \frac{\Delta}{f^i}$ **then**
6. $M_i \leftarrow M_i \cup \{v\}$ with probability $\frac{6\log n \cdot f^i}{\Delta}$
7. **for** each $v \in V$ *in parallel* **do**
8. **if** $v \in N(M_i) \setminus M_i$ **then**
9. $W_i \leftarrow W_i \cup \{v\}$
10. $V \leftarrow V \setminus (M_i \cup W_i)$
    end-for(i)
11. $I \leftarrow$ `MIS-LOWDEG`$(H[(\cup_i M_i) \cup V])$
    return I;

▶ **Lemma 1.** *At the end of Stage i, $1 \leq i \leq i^*$, with probability at least $1 - \frac{1}{n^5}$ all vertices still in V have degree at most $\frac{\Delta}{f^i}$.*

Proof. Consider a "high degree" vertex v, i.e., a vertex with degree more than Δ/f^i, at the start of Stage i. Then,

$$\Pr[v \text{ is added to } M_i \cup W_i] \geq 1 - \left(1 - \frac{6\log n \cdot f^i}{\Delta}\right)^{\frac{\Delta}{f^i}} \geq 1 - e^{-6\cdot\log n} \geq 1 - \frac{1}{n^6}.$$

Therefore, using the union bound, we see that with probability at least $1 - \frac{1}{n^5}$ all vertices in V that have degree more than Δ/f^i at the start of Stage i will join $M_i \cup W_i$ in Stage i. ◀

▶ **Lemma 2.** *Consider a Stage i, $1 \leq i \leq i^*$. With probability at least $1 - \frac{2}{n}$, the subgraph induced by M_i (i.e., $H[M_i]$) has maximum degree $12\log n \cdot f$.*

Proof. We condition on the event that all vertices that are in V at the beginning of Stage i have degree at most $\frac{\Delta}{f^{i-1}}$. For $i = 1$, this event happens with probability 1 and for $i > 1$, Lemma 1 implies that this event happens with probability at least $1 - 1/n^5$. Consider a vertex $v \in V$ that is added to M_i. Let $\deg_{M_i}(v)$ denote the degree of vertex v in $H[M_i]$. Then, $E[\deg_{M_i}(v)] \leq \frac{\Delta}{f^{i-1}} \cdot \frac{6 \log n \cdot f^i}{\Delta} = 6 \log n \cdot f$. Here we use the fact that $\deg_G(v) \leq \frac{\Delta}{f^{i-1}}$ for all $v \in V$ at the start of Stage i. Since vertices join M_i independently, using Chernoff bounds we conclude that $\Pr[\deg_{M_i}(v) \geq 12 \log n \cdot f] \leq 1/n^2$. Therefore, with probability at least $1 - 1/n$ the maximum degree of $H[M_i]$ is at most $12 \log n \cdot f$. We now drop the conditioning on the event that all vertices that are in V at the beginning of Stage i have degree at most $\frac{\Delta}{f^{i-1}}$ and use Lemma 1 and the union bound to obtain the lemma. ◀

▶ **Theorem 3.** *Algorithm* RulingSet-GG *computes a 2-ruling set of the input graph G in $O(\frac{\log \Delta}{(\log n)^\epsilon} + (\log n)^{1/2+\epsilon})$ rounds.*

Proof. It is easy to see that every stage of the algorithm runs in $O(1)$ communication rounds. Since there are i^* stages and since $i^* = O\left(\frac{\log \Delta}{(\log n)^\epsilon}\right)$, the running time of the stages all together is $O\left(\frac{\log \Delta}{(\log n)^\epsilon}\right)$. From Lemma 1 we see that the vertex set V remaining after all i^* stages induces a graph with maximum degree f with high probability. From Lemma 2 we see that the maximum degree of every $H[M_i]$ is bounded above by $O(f \cdot \log n)$ with high probability. Furthermore, since there is no interaction between any pair of M_i's and also between V and the M_i's, the maximum degree of the graph induced by $(\cup_i M_i) \cup V$ is also $O(f \cdot \log n)$. Therefore, with high probability, the MIS computation at the end of the algorithm takes $O((\log n)^{1/2+\epsilon})$ rounds using [5, Theorem 4.3]. Together these observations yield the claimed running time.

To see that I is a 2-ruling set, first observe that every vertex v ends up in $M_i \cup W_i$ for some $1 \leq i \leq i^*$ or remains in V until the end. If v ends up in W_i, it is at most 2 hops from a vertex in I that belongs to the MIS of $H[M_i]$. Otherwise, v is at most 1 hop away from a vertex in I. ◀

Using $\epsilon = 1/4$ in the above theorem results in Corollary 4. A further optimization on the choice of ϵ for graphs with degree in $2^{\omega(\sqrt{\log n})}$ is shown in Corollary 5.

▶ **Corollary 4.** *Algorithm* RulingSet-GG *computes a 2-ruling set of the input graph G in $O((\log n)^{3/4})$ rounds.*

▶ **Corollary 5.** *(i) For a graph G with $\Delta = 2^{O(\sqrt{\log n})}$, Algorithm* RulingSet-GG *computes a 2-ruling set of the input graph G in $O((\log n)^{1/2+\epsilon})$ rounds for any $\epsilon > 0$. (i) For a graph G with $\Delta = 2^{\omega(\sqrt{\log n})}$, Algorithm* RulingSet-GG *computes a 2-ruling set of the input graph G in $O((\log n)^{1/4}\sqrt{\log \Delta})$ rounds.*

Proof. We get (i) by simply plugging $\Delta = 2^{O(\sqrt{\log n})}$ into the running time expression from Theorem 3. (ii) In this case, we know that $\log \Delta = \omega(\sqrt{\log n})$ and $\log \Delta \leq \log n$. Consider the two expressions $\frac{\log \Delta}{(\log n)^\epsilon}$ and $(\log n)^{1/2+\epsilon}$ in the running time expression from Theorem 3. At $\epsilon = 0$ the first term is larger and as we increase ϵ, the first term falls and the second term increases. By the time $\epsilon = 1/4$ the second term is larger. We find a minimum value by equating the two terms and solving for ϵ. This yields an "optimal" value of

$$\epsilon = \frac{\log \log \Delta}{2 \log \log n} - \frac{1}{4}$$

and plugging this into the running time expression yields the running time bound of $O((\log n)^{1/4} \cdot \sqrt{\log \Delta})$ rounds. ◀

3 3-Ruling Sets for High Girth Graphs and Trees

Our goal in this section is to devise an $O(1)$-ruling set algorithm for high girth graphs and trees that is much faster than the 2-ruling set algorithm for general graphs from the previous section. In Algorithm RulingSet-GG we allow the graph induced by M_i to have degree as high as $O(f \cdot \log n)$ where $f = 2^{(\log n)^\epsilon}$. Computing an MIS on a graph with degree as high as this is too time consuming for our purposes. We could try to reduce f, but this will result in a corresponding increase in the number of stages. Therefore, we need to use additional ideas to help simultaneously keep the maximum degree of the graphs $H[\cup_i M_i]$ small and also the number of stages small.

Let $G = (V, E)$ be a graph with n vertices, maximum degree Δ, and girth at least 6. Let i^* be the smallest positive integer such that $\Delta^{1/2^{i^*}} \leq 6 \cdot \log n$. It is easy to check that $i^* = O(\log \log \Delta)$.

Let M_1 and M_2 be disjoints subsets of V such that the maximum vertex degree in $G[M_1]$ and in $G[M_2]$ is bounded by $O(\log n)$. We use `MIS-TWOSTAGE`(G, M_1, M_2) to denote a call to the following algorithm for computing an MIS on $G[M_1 \cup M_2]$.

1. Compute an MIS I_1 on $G[M_1]$ using the algorithm of Barenboim et al. ([5], Theorem 7.2).
2. Compute an MIS I_2 on $G[M_2 \setminus N(I_1)]$ using the algorithm of Barenboim et al. ([5], Theorem 7.2).
3. return $I_1 \cup I_2$.

This algorithm runs in $\exp(O(\sqrt{\log \log n}))$ rounds since the maximum degree in $G[M_1]$ and in $G[M_2]$ is bounded by $O(\log n)$ and therefore by Theorem 7.2 [5] each of the MIS computations requires $\exp(O(\sqrt{\log \log n}))$ rounds. If G were a tree, then we could use Theorem 7.3 in Barenboim et al. [5], which tells us that we can compute an MIS on a tree with maximum degree $O(\log n)$ in $O(\log \log n \cdot \log \log \log n)$ rounds. From this we see that a call to `MIS-TWOSTAGE`(G, M_1, M_2) runs in $O(\log \log n \cdot \log \log \log n)$ rounds when G is a tree.

In our previous algorithm, Algorithm RulingSet-GG, we used degree ranges $(\frac{\Delta}{f}, \Delta]$, $(\frac{\Delta}{f^2}, \frac{\Delta}{f}]$, etc. Here we use even larger degree ranges: $(\Delta^{1/2}, \Delta]$, $(\Delta^{1/4}, \Delta^{1/2}]$, etc. The algorithm proceeds in stages and in Stage i all vertices with degrees in the range $(\Delta^{1/2^i}, \Delta^{1/2^{i-1}}]$ are processed. To understand the algorithm and why it works consider what happens in Stage 1. (It may be helpful to consult the pseudocode of Algorithm RulingSet-HG while reading the following.) In Line 6 we allow "high degree" vertices (i.e., those with degree more than $\sqrt{\Delta}$) to join a set M_1 with a probability $\frac{6 \log n}{\Delta}$. This probability is small enough that it ensures that the expected maximum degree of the subgraph induced by M_1 is $O(\log n)$. In fact, this also holds with high probability, as shown in Lemma 8. However, as can be seen easily, there are lots of "high degree" vertices that have no neighbor in M_1. We use two ideas to remedy this situation. The first idea is to allow "low degree" vertices (i.e., those with degree at most $\sqrt{\Delta}$) also to join a set M_2, with the somewhat higher probability of $\frac{6 \log n}{\sqrt{\Delta}}$ (Line 7). This probability is low enough to ensure that the graph induced by M_2 has $O(\log n)$ maximum degree, but it is also high enough to ensure that if a "high degree" node has lots of "low degree" neighbors, it will see some neighbor in M_2, with high probability. This still leaves untouched "high degree" vertices with lots of "high degree" neighbors. To deal with these vertices, we remove not just the neighborhood of M_1, but also the 2-neighborhood of M_1. The fact that G has a high girth ensures that a "high degree" vertex that has many "high degree" neighbors has lots of vertices in its 2-neighborhood. This allows us to show that such "high degree" vertices are also removed with high probability. The above arguments are formalized in Lemma 6. We repeat this procedure for smaller degree ranges until the

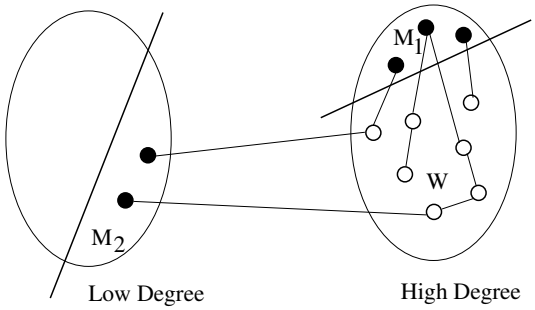


Figure 1 Figure showing one iteration of Algorithm RulingSet-HG. The figure shows the sets M_1, M_2 and W.

degree of the graph that remains is poly-logarithmic. Figure 1 shows one iteration of the algorithm. Pseudocode of our algorithm appears as Algorithm RulingSet-HG below.

Algorithm RulingSet-HG($G = (V, E)$)
1. $I \leftarrow \emptyset$
2. **for** $i = 1, 2, \cdots, i^*$ **do**
   /* Stage i ***/**
3. $M_1 \leftarrow \emptyset$; $M_2 \leftarrow \emptyset$; $W \leftarrow \emptyset$
4. **for** $v \in V$ *in parallel* **do**
5. **if** $\deg(v) > \Delta^{1/2^i}$ **then**
6. $M_1 \leftarrow M_1 \cup \{v\}$ with probability $\frac{6 \cdot \log n}{\Delta^{1/2^{i-1}}}$
   else if $\deg(v) \leq \Delta^{1/2^i}$ **then**
7. $M_2 \leftarrow M_2 \cup \{v\}$ with probability $\frac{6 \cdot \log n}{\Delta^{1/2^i}}$
8. $I \leftarrow I \cup$ `MIS-TWOSTAGE`(G, M_1, M_2)
9. **for** $v \in V \setminus (M_1 \cup M_2)$ *in parallel* **do**
10. **if** $\text{dist}(v, M_1 \cup M_2) \leq 2$ **then**
11. $W \leftarrow W \cup \{v\}$
12. $V \leftarrow V \setminus (M_1 \cup M_2 \cup W)$
   end-for(i)
13. $I \leftarrow I \cup$ `MIS`(G)
   return I;

In the following, we analyze Algorithm RulingSet-HG. We show in Lemma 6 that all nodes of degree at least $\Delta^{1/2^i}$ can be processed in the ith iteration. This is followed by Lemma 8 that argues that the degree of $G[M_1 \cup M_2]$ is $O(\log n)$, and finally Theorem 9 that shows our result for graph of girth at least 6 and trees.

▶ **Lemma 6.** *For $1 \leq i \leq i^*$, with probability at least $1 - 1/n^2$, all vertices still in V have degree at most $\Delta^{1/2^i}$ at the end of iteration i.*

Proof. Consider a vertex $v \in V$ at the start of iteration i that has degree greater than $\Delta^{1/2^i}$. Vertex v can have one of two types:
Type I : v is of Type I if at least half of v's neighbors have degree greater than $\Delta^{1/2^i}$.
Type II : v is of Type II if fewer than half of v's neighbors have degree greater than $\Delta^{1/2^i}$.

If v is of Type I, then there are at least $1/2 \cdot \Delta^{1/2^i} \cdot \Delta^{1/2^i} = \Delta^{1/2^{i-1}}/2$ vertices in v's 2-neighborhood. Here we use the fact that G has girth at least 6. Now note that any

vertex u in v's 2-neighborhood is added to $M_1 \cup M_2$ with probability at least $\frac{6\log n}{\Delta^{1/2^{i-1}}}$. Therefore, the probability that no vertex in v's 2-neighborhood is added to $M_1 \cup M_2$ is at most $(1-\frac{6\log n}{\Delta^{1/2^{i-1}}})^{|N_2(v)|}$, where $N_2(v)$ denotes the 2-neighborhood of vertex v. Here we use the fact that vertices are added to $M_1 \cup M_2$ independently. Using the lower bound $|N_2(v)| \geq \Delta^{1/2^{i-1}}/2$, we see that $\Pr[v \text{ is added to } M_1 \cup M_2 \cup W] \geq 1 - \left(1 - \frac{6\cdot\log n}{\Delta^{1/2^{i-1}}}\right)^{\frac{\Delta^{1/2^{i-1}}}{2}} \geq 1 - e^{-3\cdot\log n} = 1 - \frac{1}{n^3}$. If v is of Type II, then more than half of v's neighbors have degree less than or equal to $\Delta^{1/2^i}$. Each such "low degree" neighbor is added to M_2 with probability $6\log n/\Delta^{1/2^i}$. Therefore, $\Pr[v \text{ is added to } M_1 \cup M_2 \cup W] \geq 1 - \left(1 - \frac{6\cdot\log n}{\Delta^{1/2^i}}\right)^{\frac{\Delta^{1/2^i}}{2}} \geq 1 - e^{-3\cdot\log n} = 1 - \frac{1}{n^3}$. In either case, v is added to $M_1 \cup M_2 \cup W$ with probability at least $1 - 1/n^3$. Therefore, by the union bound every node of degree greater than $\Delta^{1/2^i}$ is added to $M_1 \cup M_2 \cup W$ with probability at least $1 - 1/n^2$. Therefore, at the end of iteration i, with probability at least $1 - 1/n^2$, there are no vertices in V with degree more than $\Delta^{1/2^i}$. ◀

▶ **Corollary 7.** *With probability at least $1 - 1/n^2$, after all i^* iterations of the for-loop in Algorithm* RulingSet-HG*, the graph G has maximum degree at most $6\log n$.*

▶ **Lemma 8.** *Consider an arbitrary iteration $1 \leq i \leq i^*$ and let $H = G[M_1 \cup M_2]$. With probability at least $1 - 2/n$, the maximum degree of a vertex in $H[M_j]$, $j = 1, 2$ is at most $12 \cdot \log n$.*

Proof. We condition on the event that all vertices that are in V at the beginning of an iteration i have degree at most $\Delta^{1/2^{i-1}}$. For $i = 1$, this event happens with probability 1 and for $i > 1$, Lemma 6 implies that this event happens with probability at least $1 - 1/n^2$. Consider a vertex $v \in V$ that is added to M_1. Let $\deg_{M_1}(v)$ denote the degree of vertex v in $G[M_1]$. Then, $E[\deg_{M_1}(v)] \leq \Delta^{1/2^{i-1}} \cdot \frac{6\cdot\log n}{\Delta^{1/2^{i-1}}} = 6 \cdot \log n$. Here we use the fact that $\deg(v) \leq \Delta^{1/2^{i-1}}$ for all $v \in V$ at the start of iteration i. Similarly, for a vertex $v \in V$ that is added to M_2, let $\deg_{M_2}(v)$ denote the degree of vertex v in $G[M_2]$. Then, $E[\deg_{M_2}(v)] \leq \Delta^{1/2^i} \cdot \frac{6\cdot\log n}{\Delta^{1/2^i}} = 6\cdot\log n$. Here we use the fact that v is added to M_2 only if $\deg(v) \leq \Delta^{1/2^i}$. Since vertices join M_1 independently, using Chernoff bounds we conclude that $\Pr[\deg_{M_1}(v) \geq 12 \cdot \log n] \leq 1/n^2$. Similarly, we conclude that $\Pr[\deg_{M_2}(v) \geq 12 \cdot \log n] \leq 1/n^2$. Therefore, with probability at least $1 - 1/n$ the maximum degree of $G[M_1 \cup M_2]$ is at most $12\log n$. We now drop the conditioning on the event that all vertices that are in V at the beginning of iteration i have degree at most $\Delta^{1/2^{i-1}}$ and use Lemma 6 and the union bound to obtain the lemma. ◀

▶ **Theorem 9.** *Algorithm* RulingSet-HG *computes a 3-ruling set of G. If G is a graph with girth at least 6 then* RulingSet-HG *terminates in $\exp(O(\sqrt{\log\log n}))$ rounds with high probability. If G is a tree then* RulingSet-HG *terminates in $O((\log\log n)^2 \cdot \log\log\log n)$ rounds with high probability.*

Proof. Consider a vertex $v \in V$ that is added to $M_1 \cup M_2 \cup W$ in some iteration i. Since the algorithm computes an MIS on $G[M_1 \cup M_2]$ and since every vertex in W is at most 2 hops (via edges in G) from some vertex in $M_1 \cup M_2$, it follows that v is at distance at most 3 from a vertex placed in I in iteration i. A vertex that is not added to $M_1 \cup M_2 \cup W$ ends up in the graph whose MIS is computed (in Line 13) and is therefore at most 1 hop away from a vertex in I. Thus every vertex in V is at most 3 hops away from some vertex in I.

The total running time of the algorithm is i^* times the worst case running time the call to the `MIS` subroutine in Line 8 plus the running time of the call to the `MIS` subroutine in Line 13. This implies that in the case of graphs of girth at least 6, Algorithm RulingSet-HG runs in

$\exp(O(\sqrt{\log\log n}))\cdot O(\log\log\Delta) = \exp(O(\sqrt{\log\log n}))$ rounds. In the case of trees, Algorithm RulingSet-HG runs in $O(\log\log\Delta\cdot\log\log n\cdot\log\log\log n) = O((\log\log n)^2\cdot\log\log\log n)$ rounds. ◀

4 Graphs with Bounded Arboricity

In the previous section, we used the fact that the absence of short cycles induces enough independence so that in each iteration, with high probability the "high degree" nodes join the set $M_1\cup M_2\cup W$. This has allowed us to process nodes of degrees in the range $(\Delta^{1/2^i},\Delta^{1/2^{i-1}}]$ in iteration i. In this section, we show that a 3-ruling set can be computed even in the presence of short cycles provided the graph has an arboricity bounded by $O(\log^k n)$ for a constant k. The algorithm we use for this case is essentially similar to that of Algorithm RulingSet-HG from Section 3. Recall from Section 3 that i^* refers to the smallest positive integer such that $\Delta^{1/2^{i^*}}\leq 6\cdot\log n$. We make the following changes to Algorithm RulingSet-HG to adapt it to graphs of arboricity $a = a(G)$.

- In iteration i, for $1\leq i\leq i^*$, a node v that has a degree at least $\Delta^{1/2^i}$ joins the set M_1 with probability $\frac{6\cdot a\log n}{\Delta^{1/2^{i-1}}}$. (See Line 6 of Algorithm RulingSet-HG.)
- In iteration i, for $1\leq i\leq i^*$, a node v with degree less than $\Delta^{1/2^i}$ joins M_2 with probability $\frac{6\cdot a\log n}{\Delta^{1/2^i}}$. (See Line 7 of Algorithm RulingSet-HG).

In the following, we show lemmas equivalent to Lemma 6 and 8 for a graph with $a\in O(\log^k n)$ for a constant k.

▶ **Lemma 10.** *Consider any iteration i for $1\leq i\leq i^*$. With probability at least $1-\frac{1}{n^2}$, all nodes still in V have degree at most $\Delta^{1/2^i}$ at the end of iteration i.*

Proof. For $i=0$, we see that each vertex has degree at most Δ with probability 1. Hence, the lemma holds for $i=0$. Let us assume inductively that the lemma holds through the first $i-1$ iterations and let us consider the ith iteration.

Consider a node v still in V at the start of iteration i that has degree at least $\Delta^{1/2^i}$. We distinguish between two cases. Recall that for a vertex v, $N_2(v)$ refers to the 2-neighborhood of v.

- v has at least half its neighbors each with degree at least $\Delta^{1/2^i}$. In this case, we notice that v has at least $\Delta^{1/2^{i-1}}/2a$ nodes at a distance of 2 from v. Otherwise, the graph induced by the set $N(v)\cup N_2(v)$ has an arboricity greater than a, which is a contradiction. Each of the vertices $u\in N_2(v)$ joins $M_1\cup M_2$ with probability at least $\frac{6\cdot a\log n}{\Delta^{1/2^{i-1}}}$. Therefore, $\Pr(v\in M_1\cup M_2\cup W)\geq 1-(1-\frac{6\cdot a\log n}{\Delta^{1/2^{i-1}}})^{\Delta^{1/2^{i-1}}/2a}\geq 1-e^{6\log n/2} = 1-1/n^3$.
- v has at most half its neighbors each with degree at least $\Delta^{1/2^i}$. In this case, each such neighbor of v joins M_2 with probability $\frac{c\cdot a\log n}{\Delta^{1/2^i}}$. Therefore, we can compute the probability that $v\in M_1\cup M_2\cup W$ as follows. $\Pr(v\in M_1\cup M_2\cup W)\geq 1-(1-\frac{6\cdot a\log n}{\Delta^{1/2^i}})^{\Delta^{1/2^i}/2a}\geq 1-e^{6\log n/2} = 1-1/n^3$.

In either case we see that v joins $M_1\cup M_2\cup W$ with a probability of $1/n^3$. Using the union bound, as in the proof of Lemma 6, vertices still in V have degree at most $\Delta^{1/2^i}$ with probability at most $1-\frac{1}{n^2}$. ◀

Lemma 8 also holds with the change that the graph $H[M_j]$ for $j=1,2$ as defined in Lemma 8 has a degree at most $12\cdot a\log n$. Since $a\in O(\log^k n)$, the above degree is in

$O(\log^{k+1} n)$, with high probability. The following theorem can be shown along the lines of Theorem 9.

▶ **Theorem 11.** *Algorithm* RulingSet-HG *computes a 3-ruling set of a graph G of arboricity $a = O(1)$ in $O((\log \log n)^3)$ rounds.*

References

1. Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.
2. L. Barenboim and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using nash-williams decomposition. In *Proc. ACM PODC*, pages 25–34, 2008.
3. L. Barenboim and M. Elkin. Distributed $(\delta + 1)$-coloring in linear (in δ) time. In *Proc. ACM STOC*, pages 111–120, 2009.
4. L. Barenboim and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *Proc. ACM PODC*, pages 410–419, 2010.
5. Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *Proc. of IEEE FOCS*, 2012, (to appear).
6. Andrew Berns, James Hegeman, and Sriram V. Pemmaraju. Super-fast distributed algorithms for metric facility location. In *Proc. ICALP(2)*, pages 428–439, 2012.
7. Beat Gfeller and Elias Vicari. A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs. In *Proc. ACM PODC*, pages 53–60, 2007.
8. K. Kothapalli, C. Scheideler, M. Onus, and C. Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ bit rounds. In *Proc. IPDPS*, 2006.
9. F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Fast deterministic distributed maximal independent set computation in growth-bounded graphs. In *Proc. of Distribtued Computing*, pages 273–287, 2008.
10. Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *CoRR*, abs/1011.5470, 2010.
11. Christoph Lenzen and Roger Wattenhofer. Mis on trees. In *Proc. ACM PODC*, pages 41–48, 2011.
12. M. Luby. A simple parallel algorithm for the maximal independent set. *SIAM Journal on Computing*, 15:1036–1053, 1986.
13. Y. Métivier, J.M. Robson, N. Saheb-Djahromi, and A. Zemmari. An optimal bit complexity randomised distributed mis algorithm. In *Proc. SIROCCO*, pages 323–337, 2009.
14. C. Nash-Williams. Decompositions of finite graphs into forests. *J. London Math*, 39(12), 1964.
15. Johannes Schneider and Roger Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proc. ACM PODC*, pages 35–44, 2008.

New bounds on the classical and quantum communication complexity of some graph properties*

Gábor Ivanyos[1], Hartmut Klauck[2], Troy Lee[3], Miklos Santha[4], and Ronald de Wolf[5]

1 **Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, Hungary**
Gabor.Ivanyos@sztaki.hu
2 **CQT and NTU Singapore**
hklauck@gmail.com
3 **CQT Singapore**
troyjlee@gmail.com
4 **CNRS - LIAFA, Université Paris Diderot, France, and CQT Singapore**
santha@liafa.univ-paris-diderot.fr
5 **CWI and University of Amsterdam, the Netherlands**
rdewolf@cwi.nl

Abstract

We study the communication complexity of a number of graph properties where the edges of the graph G are distributed between Alice and Bob (i.e., each receives some of the edges as input). Our main results are:

- An $\Omega(n)$ lower bound on the quantum communication complexity of deciding whether an n-vertex graph G is connected, nearly matching the trivial classical upper bound of $O(n \log n)$ bits of communication.
- A deterministic upper bound of $O(n^{3/2} \log n)$ bits for deciding if a bipartite graph contains a perfect matching, and a quantum lower bound of $\Omega(n)$ for this problem.
- A $\Theta(n^2)$ bound for the randomized communication complexity of deciding if a graph has an Eulerian tour, and a $\Theta(n^{3/2})$ bound for its quantum communication complexity.

The first two quantum lower bounds are obtained by exhibiting a reduction from the n-bit Inner Product problem to these graph problems, which solves an open question of Babai, Frankl and Simon [2]. The third quantum lower bound comes from recent results about the quantum communication complexity of composed functions. We also obtain essentially tight bounds for the quantum communication complexity of a few other problems, such as deciding if G is triangle-free, or if G is bipartite, as well as computing the determinant of a distributed matrix.

1998 ACM Subject Classification F.1.1 Models of Computation; F.2 Analysis of algorithms and problem complexity

Keywords and phrases Graph properties, communication complexity, quantum communication

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.148

* Most of this work was conducted at the Centre for Quantum Technologies (CQT) in Singapore, and partially funded by the Singapore Ministry of Education and the National Research Foundation. Research partially supported by the European Commission IST project Quantum Computer Science (QCS) 255961, by the CHIST-ERA project DIQIP, by Vidi grant 639.072.803 from the Netherlands Organization for Scientific Research (NWO), by the French ANR programs under contract ANR-08-EMER-012 (QRAC project) and ANR-09-JCJC-0067-01 (CRYQ project), by the French MAEE STIC-Asie program FQIC, and by the Hungarian Research Fund (OTKA, Grants K77467 and NK05645).

1 Introduction

Graphs are among the most basic discrete structures, and deciding whether graphs have certain properties (being connected, containing a perfect matching, being 3-colorable, ...) is among the most basic computational tasks. The complexity of such tasks has been studied in a number of different settings.

Much research has gone into the *query complexity* of graph properties, most of it focusing on the so-called Aandera-Karp-Rosenberg conjecture. Roughly, this says that all monotone graph properties have query complexity $\Omega(n^2)$. Here the vertex set is $[n] = \{1, \ldots, n\}$ and input graph $G = ([n], E)$ is given as an adjacency matrix whose entries can be queried. This conjecture is proved for deterministic algorithms [25], but open for randomized [13, 4].

Less—but still substantial—effort has gone into the study of the *communication complexity* of graph properties [22, 2, 12, 8]. Here the edges of G are distributed over two parties, Alice and Bob. Alice receives set of edges E_A, Bob receives set E_B (these sets may overlap), and the goal is to decide with minimal communication whether the graph $G = ([n], E_A \cup E_B)$ has a certain property. Here we obtain new bounds for the communication complexity of a number of graph properties, both in the classical and the quantum world:

- An $\Omega(n)$ lower bound on the quantum communication complexity of deciding whether G is connected, nearly matching the trivial classical upper bound of $O(n \log n)$ bits.
- Hajnal et al. [12] state as an open problem to determine the communication complexity of deciding if a bipartite graph contains a perfect matching (i.e., a set of $n/2$ vertex-disjoint edges). We prove a deterministic upper bound of $O(n^{3/2} \log n)$ bits for this, and a quantum lower bound of $\Omega(n)$.
- For deciding if a graph contains an Eulerian tour we show that the quantum communication complexity is $\Theta(n^{3/2})$ while the randomized communication complexity is $\Theta(n^2)$.

Our quantum lower bounds for the first two problems are proved by reductions from the hard inner product problem, which is $\text{IP}_n(x, y) = \sum_{i=1}^{n} x_i y_i \bmod 2$. Babai et al. [2, Section 7] showed how to reduce the disjointness problem ($\text{Disj}_n(x, y) = 1$ iff $\sum_{i=1}^{n} x_i y_i = 0$) to these graph problems, but left reductions from inner product as an open problem (they did reduce inner product to a number of other problems [2, Section 9]). In the classical world this does not make much difference since both Disj and IP require $\Omega(n)$ communication (the tight lower bound for Disj was proved only after [2] in [16]). However, in the quantum world Disj is quadratically easier than IP, so reductions from IP give much stronger lower bounds here.

While investigating the communication complexity of graph properties is interesting in its own right, there have also been applications of lower bounds for such problems. For instance, communication complexity arguments have recently been used to show new and tight lower bounds for several graph problems in distributed computing in [7]. These problems include approximation and verification versions of classical graph problems like connectivity, *s-t* connectivity, and bipartiteness. In their setting processors see only their local neighborhood in a network. Paper [7] use reductions from Disj to establish their lower bounds. Subsequently some of these results have been generalized to the case of quantum distributed computing [10], employing for instance the new reductions from IP given in this paper, which in the quantum case establish larger lower bounds than the previous reductions from Disj.

2 Preliminaries

We assume familiarity with communication complexity, referring to [18] for more details about classical communication complexity and [32] for quantum communication complexity (for information about the quantum model beyond what's provided in [32], see [21]).

Given some communication complexity problem $f : X \times Y \to R$ we use $D(f)$ to denote its classical deterministic communication complexity, $R_2(f)$ for its private-coin randomized communication complexity with error probability $\leq 1/3$, and $Q_2(f)$ for its private-coin quantum communication complexity with error $\leq 1/3$. Our upper bounds for the quantum model do not require prior shared entanglement; however, all lower bounds on $Q_2(f)$ in this paper also apply to the case of unlimited prior entanglement.

Among others we consider two well-known communication complexity problems, with $X = Y = \{0,1\}^n$ and $R = \{0,1\}$. For $x, y \in \{0,1\}^n$ we define $x \wedge y \in \{0,1\}^n$ as the bitwise AND of x and y, and $|x| = |\{i \in [n] : x_i = 1\}|$ as the Hamming weight of x.

- Inner product: $\mathrm{IP}_n(x,y) = |x \wedge y| \bmod 2$. The quantum communication complexity of this problem is $Q_2(\mathrm{IP}_n) = \Theta(n)$ [17, 5] (in fact even its *unbounded-error* quantum communication complexity is linear [11]).
- Disjointness: $\textsc{Disj}_n(x,y) = 1$ if $|x \wedge y| = 0$, and $\textsc{Disj}_n(x,y) = 0$ otherwise. Viewing x and y as the characteristic vectors of subsets of $[n]$, the task is to decide whether these sets are disjoint. It is known that $R_2(\textsc{Disj}_n) = \Theta(n)$ [16, 23] and $Q_2(\textsc{Disj}_n) = \Theta(\sqrt{n})$ [3, 1, 24]. In fact, the Aaronson-Ambainis protocol [1] can find an i such that $x_i = y_i = 1$ (if such an i exists), using an expected number of $O(\sqrt{n})$ qubits of communication. This saves a log-factor compared to the distributed implementation of Grover's algorithm in [3].

We will make use of both undirected and directed graphs. We use $\{i,j\}$ to refer to an undirected edge between vertex i and j, and (i,j) for an edge directed from i to j.

3 Reduction from Parity

We begin with a reduction from the n-bit Parity problem to the connectedness of a $2n$-vertex graph in the model of *query* complexity. This reduction was used by Dürr et al. [9, Section 8], who attribute it to Henzinger and Fredman [14]. The same reduction can be used to reduce Parity to determining if an n-by-n bipartite graph contains a perfect matching. Our hardness results for communication complexity in later sections follow by means of simple gadgets to transfer this reduction from the query world to the communication world.

▶ Claim 1. For every $z \in \{0,1\}^n$ there is a graph G_z with $2n$ vertices (where for each possible edge, its presence or absence just depends on one of the bits of z), such that if the parity of z is odd then G_z is a cycle of length $2n$, and if the parity of z is even then G_z is the disjoint union of two n-cycles.

Proof. We construct a graph G with $2n$ vertices, arranged in two rows of n vertices each. We will label the vertices as t_i and b_i for $i \in [n]$ indicating if it is in the top row or the bottom row. For $i \in [n-1]$, if $z_i = 0$ then add edges $\{t_i, t_{i+1}\}$ and $\{b_i, b_{i+1}\}$; if $z_i = 1$ then add $\{t_i, b_{i+1}\}$ and $\{b_i, t_{i+1}\}$. For $i = n$ make the same connections with vertex 1, wrapping around. See Figure 1 for illustration. If the parity of z is odd then the resulting graph G will be one $2n$-cycle, and if the parity is even then it will be two n-cycles. ◀

4 Connectivity

We first focus on the communication complexity of deciding whether a graph G is connected or not. Denote the corresponding Boolean function for n-vertex graphs by $\textsc{Connectivity}_n$ (we sometimes omit the subscript when it's clear from context). Note that it suffices for Alice and Bob to know the connected components of their graphs; additional information

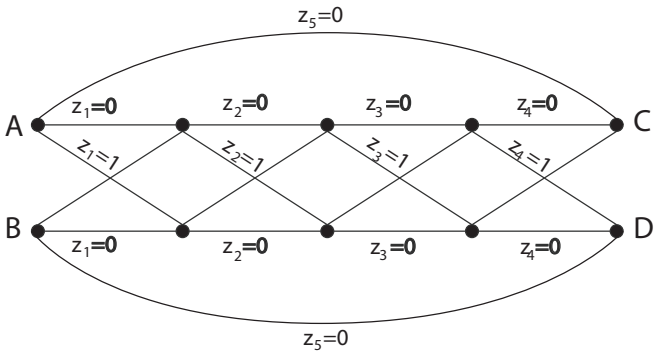


Figure 1 The string z determines the edges present in G_z. If $z_5 = 1$ there are edges connecting A with D, and B with C (omitted for clarity). When the parity of z is odd, the graph is a $2n$-cycle, and when it is even the graph is the disjoint union of two n-cycles.

about edges within their connected components is redundant for deciding connectivity. Hence the "real" input length is $O(n \log n)$ bits, which of course implies the upper bound $D(f) = O(n \log n)$. Hajnal et al. [12] showed a matching lower bound for $D(f)$. As far as we know, extending this lower bound to $R_2(\textsc{Connectivity})$ is open. The best lower bound known is $R_2(\textsc{Connectivity}) = \Omega(n)$ via a reduction from $\textsc{Disj}_n$ [2]. Since DISJ is quadratically easier for quantum communication than for classical communication, the reduction from $\textsc{Disj}_n$ only implies a quantum lower bound $Q_2(\textsc{Connectivity}) = \Omega(\sqrt{n})$.

We now improve this by a reduction from IP_n, answering an open question from [2]. Since we know $Q_2(\mathrm{IP}_n) = \Omega(n)$, this will imply $Q_2(\textsc{Connectivity}) = \Omega(n)$, which is tight up to the log-factor. We modify the graph from Claim 1 originally used in the context of query complexity to give a reduction from IP to connectivity in the communication world.

▶ **Theorem 1.** $\Omega(n) \leq Q_2(\textsc{Connectivity}_n) \leq D(\textsc{Connectivity}_n) \leq O(n \log n)$.

Proof. Let $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ be Alice and Bob's inputs, respectively. Set $z = x \wedge y$, then the parity of z is $\mathrm{IP}_n(x,y)$. We define a graph G which is a modification of the graph G_z from Claim 1 by distributing its edges over Alice and Bob, in such a way that if $\mathrm{IP}_n(x,y) = 1$ (i.e., $|z|$ is odd) then the resulting graph is a $2n$-cycle, and if $\mathrm{IP}_n(x,y) = 0$ (i.e., $|z|$ is even) then G consists of two disjoint n-cycles, and therefore is not connected. To do that we replace every edge with a "gadget" that adds two extra vertices. Formally, we will have the $2n$ vertices t_i, b_i, and $8n$ new vertices $k_i^{tt}, k_i^{bb}, k_i^{tb}, k_i^{bt}, \ell_i^{tt}, \ell_i^{bb}, \ell_i^{tb}, \ell_i^{bt}$, for $i \in [n]$. See Figure 2 for a picture of the gadgets.

We describe the gadget corresponding to the ith horizontal edge on the top. It involves the vertices $t_i, k_i^{tt}, \ell_i^{tt}, t_{i+1}$ and depends only on x_i and y_i. The gadget corresponding to the ith horizontal bottom edge is isomorphic but defined on vertices $b_i, k_i^{bb}, \ell_i^{bb}, b_{i+1}$. If $x_i = 0$ then $\{t_i, k_i^{tt}\} \in E_A$, and if $y_i = 0$ then $\{t_i, \ell_i^{tt}\} \in E_B$. Independently of the value of x_i, the edges $\{k_i^{tt}, t_{i+1}\}$ and $\{\ell_i^{tt}, t_{i+1}\}$ are in E_A. Note that this gadget is connected iff $x_i y_i = 0$.

Now we describe the gadget corresponding to the ith diagonal edge $\{t_i, b_{i+1}\}$, the gadget corresponding to $\{b_i, t_{i+1}\}$ is isomorphic to this one on the appropriate vertex set. If $x_i = 1$ then $\{t_i, \ell_i^{tb}\} \in E_A$, if $y_i = 0$ then $\{k_i^{tb}, \ell_i^{tb}\} \in E_B$, and if $y_i = 1$ then $\{\ell_i^{tb}, t_{i+1}\} \in E_B$. Finally $\{t_i, k_i^{tb}\} \in E_A$ no matter what x_i is. Note that this gadget is connected iff $x_i y_i = 1$.

In total the resulting graph G will have $10n$ vertices, and disjoint sets E_A and E_B of $O(n)$ edges. If $\mathrm{IP}_n(x,y) = 1$ then the graph consists of one cycle of length $4n$, with a

few extra vertices attached to it. If $\mathrm{IP}_n(x,y) = 0$ then the graph consists of two disjoint cycles of length $2n$ each, again with a few extra vertices attached to them. (Observe that ℓ_i^{tb} is always connected to t_i or to t_{i+1} even when $x_i = y_i = 0$). Accordingly, a protocol that can compute Connectivity on this graph computes $\mathrm{IP}_n(x,y)$, which shows $Q_2(\mathrm{IP}_n) \leq Q_2(\textsc{Connectivity}_{10n})$.

Our gadgets are slightly more complicated than strictly necessary, to ensure the sets of edges E_A and E_B are disjoint. This implies that the lower bound holds even for that special case. Note that the lower bound even holds for *sparse* graphs, as G has $O(n)$ edges. ◀

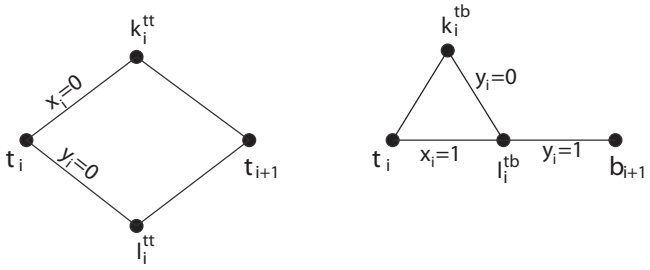


Figure 2 Two gadgets used to modify the reduction from Parity in the query complexity model to one for Inner Product in communication complexity. On the left the gadget replacing the top $z_i = 0$ edge in Fig 1; on the right the gadget for the diagonal top-to-bottom $z_i = 1$ edge.

5 Matching

The second graph problem we consider is deciding whether an $n \times n$ *bipartite* graph G contains a perfect matching. We denote this problem by $\textsc{Bipartite Matching}_n$. First, we show that the above reduction from IP can be modified to also work for Bipartite Matching.

▶ **Theorem 2.** $Q_2(\textsc{Bipartite Matching}_n) \geq \Omega(n)$.

Proof. Let $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ be respectively the inputs of Alice and Bob. As previously, we set $z = x \wedge y$, and observe again that the parity of z is $\mathrm{IP}_n(x,y)$. We go back to the query world and the $2n$-vertex graph G_z of Claim 1. Assume n is odd. Then in case the parity of z is odd, G_z is a cycle of even length $2n$ and so has a perfect matching. On the other hand, in case the parity of z is even, G_z consists of two odd cycles and so has no perfect matching.

Now we again use gadgets to transfer this idea to a reduction from inner product to matching in the communication complexity setting. For simplicity we first describe the reduction where the edge sets of Alice and Bob can overlap. We then explain a modification to make them disjoint.

The vertices of the graph G will consist of the $2n$ vertices t_i, b_i as in Figure 1 with the addition of $4n$ new vertices $k_i^t, k_i^b, \ell_i^t, \ell_i^b$ for $i \in [n]$. For every i there is a unique gadget on vertex set $\{t_i, b_i, k_i^t, k_i^b, \ell_i^t, \ell_i^b, t_{i+1}, b_{i+1}\}$. The edges $\{k_i^t, \ell_i^b\}$ and $\{k_i^b, \ell_i^t\}$ are always present in the graph, and will be given to Alice. If $x_i = 0$ then we give Alice the edges $\{t_i, t_{i+1}\}$ and $\{b_i, b_{i+1}\}$. If $y_i = 0$ we do the same thing for Bob (this is where edges may overlap). If $x_i = 1$ we give Alice the edges $\{t_i, k_i^t\}$ and $\{b_i, k_i^b\}$. If $y_i = 1$ we give Bob the edges $\{t_{i+1}, \ell_i^t\}$ and $\{b_{i+1}, \ell_i^b\}$. This is illustrated in Figure 3.

Now in case the parity of z is odd, we will have a cycle of even length, with possibly some additional disjoint edges and attached paths of length two. Thus there will be a perfect matching. In case the parity of z is even, we will have two odd cycles, and again some additional disjoint edges or attached paths of length two. Suppose, by way of contradiction, that there is a perfect matching in this case. In case $x_i y_i = 0$, this matching must include the edge $\{k_i^t, \ell_i^b\}$, since at least one of these vertices has degree one, and similarly for $\{k_i^b, \ell_i^t\}$. Thus a perfect matching in this case gives a perfect matching of two odd cycles, a contradiction. To make the edge sets disjoint, we replace horizontal edges between vertex i and $i+1$ by the gadget in the left of Figure 3. It can be seen that this does not change the properties used in the reduction. ◀

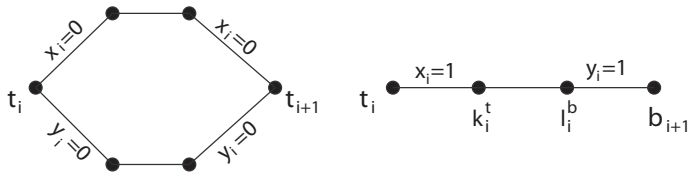


Figure 3 Two gadgets used to modify the reduction from Parity in the query complexity model to one for matching in the communication complexity model. On the right is the gadget for the top to bottom diagonal $z_i = 1$ edge. On the left, the gadget used to replace the top $z_i = 0$ horizontal edge in the graph from Figure 1 such that Alice and Bob receive disjoint sets of edges.

Second, we show a non-trivial deterministic upper bound $D(\text{Bipartite Matching}_n) = O(n^{3/2} \log n)$ by implementing a distributed version of the famous Hopcroft-Karp algorithm for finding a maximum-cardinality matching [15]. Let us first explain this algorithm in the standard non-distributed setting. The algorithm starts with an empty matching M, and in each iteration grows the size of M until it can no longer be increased. It does this by finding, in each iteration, many *augmenting paths*. An augmenting path, relative to a matching M, is a path P of odd length that starts and ends at "free" (= unmatched in M) vertices, and alternates non-matching with matching edges. Note that the symmetric difference of M and P is another matching, of size one greater than M. Each iteration of the Hopcroft-Karp algorithm does the following (using the notation of [15], we call the vertex sets of the bipartition X and Y, respectively).

1. A breadth-first search (BFS) partitions the vertices of the graph into layers. The free vertices in X are used as the starting vertices of this search, and form the initial layer of the partition. The traversed edges are required to alternate between unmatched and matched. That is, when searching for successors from a vertex in X, only unmatched edges may be traversed, while from a vertex in Y only matched edges may be traversed. The search terminates at the first layer k where one or more free vertices in Y are reached.
2. All free vertices in Y at layer k are collected into a set F. That is, a vertex v is put into F iff it ends a shortest augmenting path (i.e., one of length k). The algorithm finds a maximal set of vertex-disjoint augmenting paths of length k. This set may be computed by depth-first search (DFS) from F to the free vertices in X, using the BFS-layering to guide the search: the DFS is only allowed to follow edges that lead to an unused vertex in the previous layer, and paths in the DFS tree must alternate between unmatched and matched edges. Once an augmenting path is found that involves one of the vertices in F, the DFS is continued from the next starting vertex. After the search is finished, each of

the augmenting paths found is used to enlarge M.
The algorithm stops when a new iteration fails to find another augmenting path, at which point the current M is a maximal-cardinality matching. Hopcroft and Karp showed that this algorithm finds a maximum-cardinality matching using $O(\sqrt{n})$ iterations. Since each iteration takes time $O(n^2)$ to implement, the overall time complexity is $O(n^{5/2})$.

Now consider what happens in a distributed setting, where Alice and Bob each have some of the edges of G. In this case, one iteration of the Hopcroft-Karp algorithm can be implemented by having each party perform as much of the search as possible within their graph, and then communicate the relevant vertices and edges to the other. To be more specific, the BFS is implemented as follows. For each level, first Alice scans the vertices on the given level and lists the set of vertices which belong to the next level due to edges seen by Alice, and then Bob lists the remaining vertices of the next level. When doing a DFS, first Alice goes forward as much as possible, then Bob follows. If Bob cannot continue going forward he gives the control back to Alice who will step back. Otherwise Bob goes forward as much as he can and then gives the control back to Alice who can either step back or continue going forward. During both types of search, when a new vertex is discovered Alice or Bob communicates the vertex as well as the edge leading to the new vertex. (Note that both the BFS and the DFS give algorithms of communication cost $\Theta(n \log n)$ for the constructive version of connectivity.)

Since each vertex needs to be communicated at most once per iteration, implementing one iteration takes $O(n \log n)$ bits of communication. Since there are $O(\sqrt{n})$ iterations, the whole procedure can be implemented using $O(n^{3/2} \log n)$ bits of communication. Finding the maximum-cardinality matching of course suffices for deciding if G contains a perfect matching, so we get the same upper bound on $D(\text{Bipartite Matching}_n)$ (we don't know anything better when we allow randomization and quantum communication). We proved:

▶ **Theorem 3.** $D(\text{Bipartite Matching}_n) \leq O(n^{3/2} \log n)$.

In the usual setting of computation (not communication), Lovász [20] gave a very elegant randomized method to decide whether a bipartite graph contains a perfect matching in matrix-multiplication time. Briefly, it works as follows. The determinant of an $n \times n$ matrix A is $\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$. Thus $\det(A)$ is a degree-n polynomial in the matrix entries. Suppose we replace the nonzero entries of A_{ij} by variables x_{ij}. This turns $\det(A)$ into a polynomial $p(x)$ of degree n in (at most) n^2 variables x_{ij}. Note that the monomial $\prod_{i=1}^{n} x_{i,\sigma(i)}$ vanishes iff at least one of the $A_{i,\sigma(i)}$ equals 0. Hence a graph G has no perfect matching iff the polynomial $p(x)$ derived from its bipartite adjacency matrix A is identically equal to 0. Testing whether a polynomial p is identically equal to 0 is easy to do with a randomized algorithm: randomly choose values for the variables x_{ij} from a sufficiently large field, and compute the value of the polynomial $p(r)$. If $p \equiv 0$ then $p(r) = 0$, and if $p \not\equiv 0$ then $p(r) \neq 0$ with high probability by the Schwartz-Zippel lemma [26, 33]. Since $p(x)$ is the determinant of an $n \times n$ matrix, which can be computed in matrix-multiplication time $O(n^\omega)$,[1] we obtain the same upper bound on the time needed to decide with high probability whether a graph contains a perfect matching.

One might hope that a distributed implementation of Lovász's algorithm could improve the above communication protocol for matching, using randomization and possibly even quantum communication. Unfortunately this does not work, because it turns out that computing the determinant of an $n \times n$ matrix whose n^2 entries are distributed over Alice and

[1] The current best bound is $\omega \in [2, 2.373)$ [6, 27, 31].

Bob, takes $\Omega(n^2)$ qubits of communication. In fact, even deciding whether the determinant equals 0 modulo 2 takes $\Omega(n^2)$ qubits of communication. We show this by a reduction from IP_{n^2}. Let DET_n be the communication problem where Alice is given an n-by-n Boolean matrix X, Bob an n-by-n Boolean matrix Y, and the desired output is $\det(X \wedge Y)$, where $X \wedge Y$ is the bitwise AND of X and Y.

▶ **Theorem 4.** $\Omega(n^2) \leq Q_2(\mathrm{DET}_n)$.

Proof. As before, we first explain a reduction in the query world from Parity of n^2 bits to computing the determinant of a $(2n+2) \times (2n+2)$ matrix. The basic idea of the proof goes back to Valiant [30]. Say that we want to compute the parity of the bits of an n^2-bit string Z, and arrange the bits of Z into an n-by-n matrix. We construct a directed bipartite graph G_Z with $2n+2$ vertices, $n+1$ on each side (we will refer to these as left-hand side and right-hand side). Label the vertices on the left-hand side as t and ℓ_i for $i \in [n]$, and those on the right-hand side as s and r_i for $i \in [n]$. For every $i \in [n]$, we add the edges (s, ℓ_i) and (r_i, t). For every (i,j) with $Z(i,j) = 1$ we put an edge (ℓ_i, r_j). Finally we put the edge (t,s), and self-loops are added to all vertices but s and t.

▶ Claim 2. $\det(G_Z) = -|Z|$.

Proof. Note that $\det(G_Z) = \sum_\sigma (-1)^{\chi(\sigma)} \prod_i G_Z(i, \sigma(i))$. Consider a permutation that contributes to this sum. In this case, $\sigma(\ell_i) = r_j$ for some i,j for which $Z(i,j) = 1$. We then must have $\sigma(r_j) = t, \sigma(t) = s, \sigma(s) = \ell_i$ and that σ fixes all other vertices. The sign of σ is negative, and we get such a contribution for every i,j such that $Z(i,j) = 1$. ◀

Now again we transfer this reduction to the communication complexity setting by means of a gadget. Say that Alice has X, an n-by-n matrix and similarly Bob has Y and they want to compute $|X \wedge Y| \bmod 2$. We will actually count the number of zeros in $X \wedge Y$, which clearly then allows us to know the number of ones and so the parity.

We give Alice the set of edges E_A and Bob the set of edges E_B. Unlike in the previous reductions, in this case E_A and E_B will not be disjoint (we do not know how to do the reduction with disjoint E_A, E_B). Put $(s, \ell_i), (\ell_i, \ell_i) \in E_A$ for all $i \in [n]$ and similarly $(r_i, t), (r_i, r_i) \in E_B$ for all $i \in [n]$. For all (i,j) where $X(i,j) = 0$ put $(\ell_i, r_j) \in E_A$, and similarly for all (i,j) where $Y(i,j) = 0$ put $(\ell_i, r_j) \in E_B$. Thus in $E_A \cup E_B$ there is an edge (ℓ_i, r_j) if and only if $X(i,j)Y(i,j) = 0$. Thus by Claim 2 from the determinant of the graph with edges $E_A \cup E_B$ we can determine the number of zeros in $X \wedge Y$. ◀

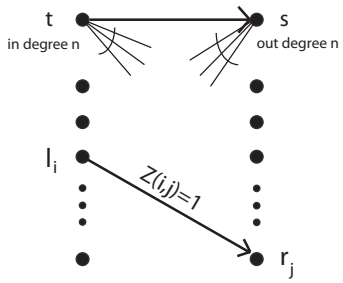


Figure 4 The construction of the graph G_Z. Self-loops omitted for clarity.

In fact what our proof shows is that even computing the determinant over $\mathbb{F}_2$ already requires $\Omega(n^2)$ qubits of communication. Independently of our work, Sun and Wang [28]

recently proved a stronger result: for every prime p, deciding singularity over the finite field $\mathbb{F}_p$ requires $\Omega(n^2 \log p)$ qubits of communication. Their proof is substantially more complicated than ours.

6 Eulerian tour

An *Eulerian tour* in a graph G is a cycle that goes through each edge of the graph exactly once. A well-known theorem of Euler states that G has such a tour iff it is connected and all its vertices have even degree. Denote the corresponding communication complexity problem for n-vertex graphs by $\textsc{Euler}_n$. Note that when the sets E_A and E_B are allowed to overlap, deciding if the degree $\deg(v)$ of a fixed vertex $v \in [n]$ is even is essentially equivalent to IP_{n-1}, as follows. Let $x \in \{0,1\}^{n-1}$ be the characteristic vector of the neighbors of v in E_A, and $y \in \{0,1\}^{n-1}$ the same for E_B, then we have $\deg(v) = |x \vee y| = |x| + |y| - |x \wedge y|$. Since Alice and Bob can send each other the numbers $|x|$ and $|y|$ using a negligible $\log n$ bits, computing $deg(v)$ mod 2 is essentially equivalent to computing $|x \wedge y| \bmod 2 = \mathrm{IP}_{n-1}(x,y)$.

Now we show how to embed into $\textsc{Euler}_{3n+4}$ an OR_n of disjoint IP_n's. As usual, we first explain the reduction in the query world. For $i \in [n]$, let $z^i \in \{0,1\}^n$, and suppose that we want to compute $\mathrm{OR}_n(|z^1| \bmod 2, \ldots, |z^n| \bmod 2)$. We construct a graph G with $n+2$ left vertices ℓ_i and $n+2$ right vertices r_i for $0 \le i \le n+1$, and n middle vertices m_i for $i \in [n]$. Independently from the strings z^i, the graph G always has the edges $\{\ell_i, \ell_{i+1}\}$ and $\{r_i, r_{i+1}\}$ for $0 \le i \le n$ and the edges $\{m_i, m_{i+1}\}$ for $1 \le i \le n-1$. It also contains the following 5 edges: $\{\ell_0, m_1\}, \{r_0, m_1\}, \{\ell_{n+1}, m_n\}, \{r_{n+1}, m_n\}, \{m_1, m_n\}$. We call these edges *fixed* edges. Finally, for every (i,j) with $z^i_j = 1$ we add the edges $\{\ell_i, m_j\}$ and $\{r_i, m_j\}$. Observe that G is already connected by the fixed edges. See Figure 5 for an illustration.

▶ Claim 3. G is Eulerian if and only if $\mathrm{OR}_n(|z^1| \bmod 2, \ldots, |z^n| \bmod 2) = 0$.

Proof. In the subgraph restricted to the fixed edges every vertex has even degree. Therefore we can restrict our attention to the degrees with respect to the remaining edges that depend on the values z^i_j. All the middle vertices have even degrees since for all (i,j), we add 0 or 2 edges adjacent to m_j. For every $i \in [n]$, the degrees of ℓ_i and r_i are the same since we add the edge $\{\ell_i, m_j\}$ exactly when we add the edge $\{r_i, m_j\}$. The degree of ℓ_i is the Hamming weight of z^i. Therefore G is Eulerian iff $|z^i|$ is even for all $i \in [n]$. ◀

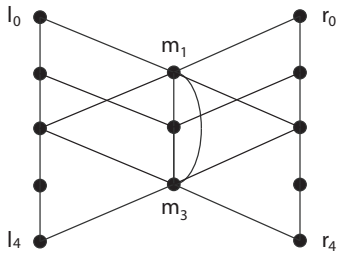


Figure 5 Illustration of the graph to reduce OR of parities to Eulerian tour in the query model. In this example, $n = 3$ and $z^1 = 010, z^2 = 101, z^3 = 000$.

The transfer of this reduction to the communication complexity setting is quite simple. Suppose that for each $i \in [n]$ Alice has string $x^i \in \{0,1\}^n$, and Bob has $y^i \in \{0,1\}^n$, and they want to compute the function $\mathrm{OR}_n(\mathrm{IP}_n(x^1, y^1), \ldots, \mathrm{IP}_n(x^n, y^n))$. Let us suppose that

n is even, then $\mathrm{IP}_n(x^i, y^i) = \sum_j (\bar{x}^i_j \vee \bar{y}^i_j) \bmod 2$. For all (i,j) such that $x^i_j = 0$ we put the edges $\{\ell_i, m_j\}$ and $\{r_i, m_j\}$ in E_A, and similarly, for $y^i_j = 0$ we put the edges $\{\ell_i, m_j\}$ and $\{r_i, m_j\}$ in E_B. Thus in $E_A \cup E_B$ the edges $\{\ell_i, m_j\}$ and $\{r_i, m_j\}$ exist iff $\bar{x}^i_j \vee \bar{y}^i_j = 1$. Therefore, by Claim 3 $\mathrm{OR}_n(\mathrm{IP}_n(x^1, y^1), \ldots, \mathrm{IP}_n(x^n, y^n)) = 0$ iff G is Eulerian.

We can easily reduce $\textsc{Disj}_{n^2}$ on n^2-bit instances with intersection size 0 or 1 to $\mathrm{OR}_n \circ \mathrm{IP}_n$. Since even that special case of $\textsc{Disj}_{n^2}$ requires linear classical communication [23], we obtain a tight lower bound $R_2(\textsc{Euler}_n) = \Omega(n^2)$.

The quantum communication complexity of $\mathrm{OR}_n(\mathrm{IP}_n(x^1, y^1), \ldots, \mathrm{IP}_n(x^n, y^n))$ is $\Omega(n^{3/2})$. This follows because for any $f(g(x^1, y^1), \ldots, g(x^n, y^n))$ where g is strongly balanced (meaning that all rows and columns in the communication matrix $M(x,y) = (-1)^{g(x,y)}$ sum to zero), the quantum communication complexity of f is at least the approximate polynomial degree of f, times the discrepancy bound of g [19, Cor. 3]. In our case, OR_n has approximate degree $\Omega(\sqrt{n})$ and IP_n contains a 2^{n-1}-by-2^{n-1} strongly balanced submatrix with discrepancy bound $\Omega(n)$. Thus we get $Q_2(\textsc{Euler}_n) \geq Q_2(\mathrm{OR}_n \circ \mathrm{IP}_n) \geq \Omega(n^{3/2})$.

This quantum lower bound is in fact tight: we first decide if G is connected using $O(n \log n)$ bits of communication (Section 4), and if so then we use the Aaronson-Ambainis protocol to search for a vertex of odd degree (deciding whether a given vertex has odd degree can be done deterministically with $O(n)$ bits of communication). Thus we have:

▶ **Theorem 5.** *$R_2(\textsc{Euler}_n) = \Theta(n^2)$ and $Q_2(\textsc{Euler}_n) = \Theta(n^{3/2})$.*

7 Other problems

In this section we look at the quantum and classical communication complexity of a number of other graph properties. Most results here are easy observations based on previous work, but worth making nonetheless.

Suppose we want to decide whether G contains a triangle. Papadimitriou and Sipser [22, pp. 266–7][2] gave a reduction from $\textsc{Disj}_m$ to $\textsc{Triangle}_n$ for $m = \Omega(n^2)$, which implies $R_2(\textsc{Triangle}_n) = \Theta(n^2)$. Since we know that $Q_2(\textsc{Disj}_m) = \Theta(\sqrt{m})$, it also follows that $Q_2(\textsc{Triangle}_n) = \Omega(n)$.

This quantum lower bound is actually tight, which can be seen as follows. First Alice checks if there already is a triangle within the edges E_A, and Bob does the same for E_B. If not, then Alice defines the set of edges $S_A = \{(a,b) \mid \exists\ c\ s.t.\ (a,c), (b,c) \in E_A\}$ which would complete a triangle for her, and uses the Aaronson-Ambainis protocol to try to find one among Bob's edges (i.e., she searches for an edge in $S_A \cap E_B$). Since $|S_A| \leq \binom{n}{2}$, this process will find a triangle if Alice already holds two of its edges, using $O(n)$ qubits of communication. Bob does the same from his perspective. If G contains a triangle, then either Alice or Bob has at least two edges of this triangle. Hence this protocol will find a triangle with high probability if one exists, using $O(n)$ qubits of communication. Thus we have:

▶ **Theorem 6.** *$R_2(\textsc{Triangle}_n) = \Theta(n^2)$ and $Q_2(\textsc{Triangle}_n) = \Theta(n)$.*

Deterministic protocols can decide whether a given graph G is bipartite using $O(n \log n)$ bits of communication, as follows. Being bipartite is equivalent to being 2-colorable. Alice starts with some vertex v_1, colors it red, and colors all of its neighbors (within E_A) blue. Then she communicates all newly-colored vertices and their colors to Bob. Bob continues

[2] Word of warning: Papadimitriou and Sipser [22] use the term "inner product" for what is now commonly called the "intersection problem," i.e., the negation of disjointness.

coloring the neighbors of v_1 blue, and once he's done he communicates the newly-colored vertices and their colors to Alice. If all vertices have been colored then Alice stops, otherwise she chooses an uncolored neighbor v_2 of a blue vertex, colors v_2 red, and continues as above coloring v_2's neighbors blue. A connected graph is 2-colorable iff this process terminates without encountering a vertex colored both red and blue (if the graph is not connected then Alice and Bob can treat each connected component separately). Since each vertex will be communicated at most once, the whole process takes $O(n \log n)$ bits.

Babai et al. [2, Section 9] state a reduction from IP_n to bipartiteness (see also [29] for details of such a reduction), which implies a nearly-matching quantum lower bound $Q_2(\textsc{Bipartiteness}_n) = \Omega(n)$.

▶ **Theorem 7.** $\Omega(n) \leq Q_2(\textsc{Bipartiteness}_n) \leq D(\textsc{Bipartiteness}_n) \leq O(n \log n)$.

8 Conclusion and open problems

We studied the communication complexity (quantum and classical) of a number of natural graph properties, obtaining nearly tight bounds for many of them. Some open problems:

- For $\textsc{Connectivity}_n$, can we improve the quantum upper bound from the trivial $O(n \log n)$ to $O(n)$, matching the lower bound? One option would be to run a distributed version of the $O(n)$-query quantum algorithm of Dürr et al. [9], but this involves a classical preprocessing phase that seems to require $O(n \log n)$ communication. Another option would be to run some kind of quantum random walk on the graph, starting from a random vertex, and test whether it converges to a superposition of all vertices.
- For Bipartite Matching, can we show that the deterministic $O(n^{3/2} \log n)$-bit protocol is essentially optimal, for instance by means of a $2^{\Omega(n^{3/2})}$ lower bound on the rank of the associated communication matrix? Can we improve this upper bound using randomization and/or quantum communication, possibly matching the $\Omega(n)$ lower bound?
- Can we extend the $D(\textsc{Bipartite Matching}_n) \leq O(n^{3/2} \log n)$ bound to general graphs?

Acknowledgements We thank Rahul Jain for several insightful discussions.

References

1 S. Aaronson and A. Ambainis. Quantum search of spatial regions. *Theory of Computing*, 1(1):47–79, 2005. Earlier version in FOCS'03. quant-ph/0303041.

2 L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity theory. In *Proceedings of 27th IEEE FOCS*, pages 337–347, 1986.

3 H. Buhrman, R. Cleve, and A. Wigderson. Quantum vs. classical communication and computation. In *Proceedings of 30th ACM STOC*, pages 63–68, 1998. quant-ph/9802040.

4 A. Chakrabarti and S. Khot. Improved lower bounds on the randomized complexity of graph properties. *Random Structures and Algorithms*, 30(3):427–440, 2007. Earlier in ICALP'01.

5 R. Cleve, W. van Dam, M. Nielsen, and A. Tapp. Quantum entanglement and the communication complexity of the inner product function. In *Proceedings of 1st NASA QCQC conference*, volume 1509 of *LNCS*, 61–74. Springer, 1998. quant-ph/9708019.

6 D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. Earlier version in STOC'87.

7 A. DasSarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proceedings of 43rd ACM STOC*, pages 363–372, 2011.

8 P. Duris and P. Pudlák. On the communication complexity of planarity. In *Proceedings of 7th Fundamentals of Computation Theory (FCT'89)*, pages 145–147, 1989.

9 C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. Earlier in ICALP'04.

10 M. Elkin, H. Klauck, D. Nanongkai, and G. Pandurangan. Quantum distributed network computing: Lower bounds and techniques. Manuscript, 2012.

11 J. Forster. A linear lower bound on the unbounded error probabilistic communication complexity. In *Proc. of 16th IEEE Conf. on Computational Complexity*, 100–106, 2001.

12 A. Hajnal, W. Maass, and G. Turán. On the communication complexity of graph properties. In *Proceedings of 20th ACM STOC*, pages 186–191, 1988.

13 P. Hajnal. An $n^{4/3}$ lower bound on the randomized complexity of graph properties. *Combinatorica*, 11:131–143, 1991. Earlier version in Structures'90.

14 M. R. Henzinger and M. L. Fredman. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22(3):351–362, 1998.

15 J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. Earlier version in FOCS'71.

16 B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

17 I. Kremer. Quantum communication. Master's thesis, Hebrew University, CS Dept., 1995.

18 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge Univ Press, 1997.

19 T. Lee and S. Zhang. Composition theorems in communication complexity. In *Proceedings of the 37th ICALP*, pages 475–489, 2010. arXiv:1003.1443.

20 L. Lovász. On determinants, matchings, and random algorithms. In *Proceedings of 2nd Fundamentals of Computation Theory (FCT'79)*, pages 565–574, 1979.

21 M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

22 C. H. Papadimitriou and M. Sipser. Communication complexity. *Journal of Computer and System Sciences*, 28(2):260–269, 1984. Earlier version in STOC'82.

23 A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

24 A. Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya of the Russian Academy of Sciences, mathematics*, 67(1):159–176, 2003. quant-ph/0204025.

25 R. Rivest and S. Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3:371–384, 1976.

26 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.

27 A. Stothers. *On the Complexity of Matrix Multiplication*. PhD thesis, University of Edinburgh, 2010.

28 X. Sun and C. Wang. Randomized communication complexity for linear algebra problems over finite fields. In *Proceedings of 29th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2012)*, pages 477–488, 2012.

29 X. Sun, C. Wang, and W. Yu. The relationship between inner product and counting cycles. In *Proceedings of LATIN'2012*, pages 643–654, 2012.

30 L. Valiant. Completeness classes in algebra. In *Proc. of 11th ACM STOC*, 249–261, 1979.

31 V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of 44th ACM STOC*, pages 887–898, 2012.

32 R. de Wolf. Quantum communication and complexity. *Theoretical Computer Science*, 287(1):337–353, 2002.

33 R. E. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of EUROSAM 79*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226, 1979.

On Bisimilarity of Higher-Order Pushdown Automata: Undecidability at Order Two

Christopher Broadbent[*1] **and Stefan Göller**[† 2]

1 **Université Paris Diderot-Paris 7 and CNRS**
Paris, France
christopher.broadbent@univ-paris-diderot.fr

2 **Université Paris Diderot-Paris 7 and CNRS / University of Bremen**
Paris, France / Bremen, Germany
goeller@informatik.uni-bremen.de

Abstract

We show that bisimulation equivalence of order-two pushdown automata is undecidable. Moreover, we study the *lower order problem* of higher-order pushdown automata, which asks, given an order-k pushdown automaton and some $k' < k$, to determine if there exists a reachable configuration that is bisimilar to some order-k' pushdown automaton. We show that the lower order problem is undecidable for each $k \geq 2$ even when the input k-PDA is deterministic and real-time.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Bisimulation equivalence, Higher-order pushdown automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.160

1 Introduction

Among the various notions of equivalence in concurrency theory [29] bisimulation equivalence (bisimilarity for short) is undoubtedly the central one in formal verification. For instance, elegant characterizations of the bisimulation-invariant fragments of well-known logics like first-order logic, monadic second-order logic or monadic path logic have been obtained in terms of modal logic [28], the modal μ-calculus [9], and CTL* [17], respectively.

The resulting decision problem, given two transition systems and a state of each of them, to decide whether the two states are bisimilar, is well-known to be complete for deterministic polynomial time on finite systems [2]. The status of decidability and of computational complexity of bisimilarity on infinite state systems is significantly less clear. A prominent such example is the class of systems described by pushdown automata (pushdown systems): Decidability was proven by Sénizergues [21], extending his famous decidability result on language equivalence of deterministic pushdown automata [20] (see [25] for a primitive recursive upper bound and [11] for a recent proof); however, the best known lower bound is EXPTIME [15]. Unfortunately, there are only few classes of infinite state systems, where bisimilarity is decidable and the precise complexity is known [10, 3].

* The author was supported by La Fondation Sciences Mathématiques de Paris. This work was also supported by by AMIS (ANR 2010 JCJC 0203 01 AMIS), FREC (ANR 2010 BLAN 0202 02 FREC) and VAPF (Région IdF)

† The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 160–172

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

Worse still, bisimilarity is not known to be decidable on PA-processes, ground tree rewrite systems, nor on higher-order pushdown systems, whereas its weaker variant (weak bisimilarity) is not known to be decidable on BPP nor on BPA, just to mention some prominent examples. We refer to [23] for an up-to-date record on the decidability and complexity status of bisimilarity of infinite state systems in Mayr's Process Rewrite Systems hierarchy.

A milestone technique for proving lower bounds for bisimilarity on infinite state systems entitled "Defender's Forcing" has been introduced by Jančar and Srba [12]. Bisimulation equivalence can be seen as a game between "Attacker" and "Defender", where Defender wins the game if and only if the pair of states are bisimilar. When aiming at showing hardness of bisimilarity via reduction from a hard problem, the simple but powerful idea of "Defender's Forcing" is to construct, whenever necessary, a subgame that allows Defender to make a choice. This is important since, in a certain sense, Attacker has much more freedom in the bisimulation game, as in each round he is the one who chooses an outgoing transition of one of the two states, whereas Defender has to respond with a matching transition in the other system. With this technique lower bounds on various classes of infinite systems have been proven, for instance Σ_1^1-completeness of bisimilarity on prefix-recognisable graphs [12].

A further natural question is to decide whether a given infinite system is bisimilar to a finite one, the so-called *regularity* problem. It seems that even less is known about this problem, for instance, decidability is open for any class of systems that lies between pushdown systems (resp. PA-processes) and Mayr's class of Process Rewrite Systems [23]. However for deterministic pushdown automata decidability follows from [27, 24].

Higher-order pushdown automata were introduced by Maslov [16] and independently by Damm and Goerdt [7]. They generate the same class of trees as *safe* higher order recursion schemes [14], which have applications to software verification for higher-order programs.

To the best of the authors' knowledge the decidability status of bisimilarity on higher-order pushdown automata is open so far, but not explicitly stated as such in the literature. In our first result we show that bisimilarity of higher-order pushdown automata is already undecidable at order two. Inspired by [12] we show undecidability via a reduction from the infinitary version of the Modified Post's Correspondence Problem (MPCP). We must, however, restrict ourselves to instances of the infinitary MPCP for which all homomorphisms are non-erasing for our proofs to work (in contrast to [12]). To obtain this result, we use the above-mentioned technique "Defender's Forcing". Our undecidability result confirms that to some extent the decidability of bisimilarity of equational graphs with finite out-degree [21] cannot be significantly improved. It is worth mentioning that transition graphs of higher-order pushdown automata have finite out-degree and decidable monadic second-order theories [4, 5]. Deciding equivalence of deterministic order-k pushdown automata is an interesting open problem, although some progress has been made on this by Stirling [26].

In the second part of the paper, we study the *lower order* problem, which asks, given some configuration c of some order-k pushdown automaton and some $k' < k$, whether c can reach a configuration that is bisimilar to some configuration of some order-k' pushdown automaton. When $k' = k-1$ this question can be seen to ask whether the order-k pushdown automaton always exhibits behaviour that is 'inherently order-k'. When $k' = 0$, it is a weaker variant of the regularity problem, which asks whether a k-PDA is bisimilar to some finite state system. The property queried by the latter implies the property queried by the 'lower order problem' with $k' = 0$, but not *vice versa*.

We show that the lower order problem is undecidable whenever $k \geq 2$ even when the input order-k pushdown automaton is deterministic and real-time (i.e. free of ε-transitions).

2 Preliminaries

Transition systems and bisimulation equivalence. By $\mathbb{N} \stackrel{\text{def}}{=} \{1, 2, \ldots\}$ we denote the *naturals*. For each $n \in \mathbb{N}$ we define $[1, n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Let us fix a countable set of *actions* $\mathbb{A}$. A *labeled transition system* is a tuple $\mathcal{T} = (S, \mathsf{A}, \{\xrightarrow{a} \mid a \in \mathsf{A}\})$, where S is a set of *states*, $\mathsf{A} \subseteq \mathbb{A}$ is a finite set of *actions*, and $\xrightarrow{a} \subseteq S \times S$ is a binary *transition relation* for each $a \in \mathsf{A}$. A state $s \in S$ is called *deterministic* if for each $a \in \Sigma$ we have $|\{s' \in S \mid s \xrightarrow{a} s'\}] \leq 1$. We say $\mathcal{T}$ is *deterministic* if every state of $\mathcal{T}$ is deterministic. We naturally extend the transition relation $\xrightarrow{a}$ inductively to words, as follows: $\xrightarrow{\varepsilon} \stackrel{\text{def}}{=} \{(s, s) \mid s \in S\}$ and $\xrightarrow{wa} \stackrel{\text{def}}{=} \{(s, t) \mid \exists u \in S : s \xrightarrow{w} u \text{ and } u \xrightarrow{a} t\}$. We write $s \xrightarrow{w}$ in case $s \xrightarrow{w} s'$ for some state s', for each word $w \in \Sigma^*$. A binary relation $R \subseteq S \times S$ is called *bisimulation* if for each $(s_1, s_2) \in R$, for each $a \in \mathsf{A}$ and for each $i \in \{1, 2\}$ we have that if $s_i \xrightarrow{a} s_i'$ for some $s_i' \in S$, then $s_{3-i} \xrightarrow{a} s_{3-i}'$ for some $s_{3-i}' \in S$ such that $(s_1', s_2') \in R$.

We write $s_1 \sim s_2$ if there is some bisimulation R with $(s_1, s_2) \in R$. It is also sometimes useful to think of bisimulation equivalence as a game played between *Attacker* and *Defender*. Starting from a pair of states (s_1, s_2) the game proceeds in rounds in which Attacker makes the first move by choosing some label a and some transition $s_i \xrightarrow{a} s_i'$ for some $i \in \{1, 2\}$. Defender has to respond with some transition $s_{3-i} \xrightarrow{a} s_{3-i}'$ and the new game position is (s_1', s_2'). If one of the players cannot make an appropriate move, then the other player wins, and moreover Defender wins every infinite game.

Higher-order pushdown automata. Let us inductively define the set of *k-stacks*, for each $k \geq 1$, over some finite stack alphabet Γ with $[,] \notin \Gamma$ and where $\bot \notin \Gamma$ is a special *bottom-of-stack symbol*:

- A 1-*stack* is an element of $\Gamma^* \bot$.
- A $(k+1)$-*stack* is a finite sequence $[\alpha_1][\alpha_2] \cdots [\alpha_n]$, where $n \geq 1$ and α_i is a k-stack for each $i \in [1, n]$.

Let us denote by $\mathsf{Stacks}_k(\Gamma)$ the set of all *k-stacks* over Γ. The *empty order k-stack* $\bot_k$ is inductively defined as $\bot_1 \stackrel{\text{def}}{=} \bot$ and $\bot_{k+1} \stackrel{\text{def}}{=} [\bot_k]$ for each $k \in \mathbb{N}$.

Over each 1-stack α we define the (partial) operation swap_w for each $w \in \Gamma^* \cup \Gamma^* \bot$ as

$$\mathsf{swap}_w(\alpha) \stackrel{\text{def}}{=} \begin{cases} wa_2 \cdots a_n & \text{if } w \in \Gamma^*,\ \alpha = a_1 \cdots a_n \bot,\ n \geq 1 \text{ and } a_i \in \Gamma \text{ for each } i \in [1, n], \\ w & \text{if } w \in \Gamma^* \bot \text{ and } \alpha = \bot, \text{ and} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\text{and} \quad \mathsf{top}_1(\alpha) \stackrel{\text{def}}{=} \begin{cases} a_1 & \text{if } \alpha = a_1 \cdots a_n \bot \text{ with } n \geq 1 \text{ and } a_i \in \Gamma \text{ for each } i \in [1, n] \text{ and} \\ \bot & \text{otherwise.} \end{cases}$$

Let us define the partial operation $\mathsf{pop}_1(\alpha) \stackrel{\text{def}}{=} \mathsf{swap}_\varepsilon(\alpha)$ and for each k-stack $\alpha =$

$[\alpha_1][\alpha_2]\cdots[\alpha_n]$ with $k \geq 2$ let us define:

$$
\begin{array}{rcl}
\mathsf{swap}_w(\alpha) & \stackrel{\text{def}}{=} & [\mathsf{swap}_w(\alpha_1)][\alpha_2]\cdots[\alpha_n] \\
\mathsf{push}_k(\alpha) & \stackrel{\text{def}}{=} & [\alpha_1][\alpha_1][\alpha_2]\cdots[\alpha_n] \\
\mathsf{push}_\ell(\alpha) & \stackrel{\text{def}}{=} & [\mathsf{push}_\ell(\alpha_1)][\alpha_2]\cdots[\alpha_n] \quad \text{for each } 2 \leq \ell < k \\
\mathsf{pop}_k(\alpha) & \stackrel{\text{def}}{=} & \begin{cases} [\alpha_2]\cdots[\alpha_n] & \text{if } n \geq 2 \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathsf{pop}_\ell(\alpha) & \stackrel{\text{def}}{=} & [\mathsf{pop}_\ell(\alpha_1)][\alpha_2]\cdots[\alpha_n] \quad \text{for each } 2 \leq \ell < k \\
\mathsf{top}_k(\alpha) & \stackrel{\text{def}}{=} & \alpha_1 \\
\mathsf{top}_\ell(\alpha) & \stackrel{\text{def}}{=} & \mathsf{top}_\ell(\alpha_1) \quad \text{for each } 1 \leq \ell < k
\end{array}
$$

Let $\mathsf{Op}_k \stackrel{\text{def}}{=} \{\mathsf{swap}_w \mid w \in \Gamma^* \cup \Gamma^*\bot\} \cup \{\mathsf{pop}_\ell \mid \ell \in [1,k]\} \cup \{\mathsf{push}_\ell \mid \ell \in [2,k]\}$ denote the set of *k-operations*. Note $\alpha \in \mathsf{Stacks}_k$ and $\mathsf{op} \in \mathsf{Op}_k$ implies $\mathsf{op}(\alpha) \in \mathsf{Stacks}_k$ if $\mathsf{op}(\alpha)$ is defined.

For each $k \geq 1$, an *order-k pushdown automaton (k-PDA)* is a tuple $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$, where Q is a finite set of *control locations*, $\mathsf{A} \subseteq \mathbb{A}$ is a finite set of actions, Γ is a finite *stack alphabet*, and where $\Delta \subseteq Q \times (\Gamma \cup \{\bot\}) \times \mathsf{A} \times Q \times \mathsf{Op}_k$ is a finite set of *stack rewrite rules*, where each $(q, x, a, q, \mathsf{op}) \in \Delta$ satisfies (i) $x = \bot$ and $\mathsf{op} = \mathsf{swap}_w$ implies $w \in \Gamma^*\bot$ and (ii) $x \in \Gamma$ and $\mathsf{op} = \mathsf{swap}_w$ implies $w \in \Gamma^*$. We abbreviate $(q, x, a, q', \mathsf{op}) \in \Delta$ by $qx \overset{a}{\hookrightarrow}_{\mathcal{P}} q'\mathsf{op}$.

The transition system of $\mathcal{P}$ is $\mathcal{T}(\mathcal{P}) \stackrel{\text{def}}{=} (Q \times \mathsf{Stacks}_k(\Gamma), \mathsf{A}, \{\overset{a}{\longrightarrow}_{\mathcal{P}} \mid a \in \mathsf{A}\})$, where $(q,\alpha) \overset{a}{\longrightarrow}_{\mathcal{P}} (q',\alpha')$ if there is $qx \overset{a}{\hookrightarrow}_{\mathcal{P}} q'\mathsf{op}$ in Δ such that $\mathsf{top}_1(\alpha) = x$ and $\alpha' = \mathsf{op}(\alpha)$ for each $q, q' \in Q$, each $a \in \mathsf{A}$ and each $\alpha, \alpha' \in \mathsf{Stacks}_k$. Thus, states of $\mathcal{T}(\mathcal{P})$ are elements of $Q \times \mathsf{Stacks}_k(\Gamma)$ that we also denote as *configurations of $\mathcal{P}$*. We call a configuration (q_0, α_0) of $\mathcal{P}$ *normed* if there exists some control location $q_f \in Q$ with $(q_f, \bot_k) \not\overset{a}{\longrightarrow}$ (emits no a-transition) for each $a \in \Sigma$, and such that every configuration (q,α) with $(q_0,\alpha_0) \longrightarrow^* (q,\alpha)$ we have $(q,\alpha) \longrightarrow^* (q_f, \bot_k)$, where $\longrightarrow^*$ is the reflexive transitive closure of $\bigcup_{a\in\Sigma} \overset{a}{\longrightarrow}$.
We can now state the decision problem which we study in the next section.

k-PDA-Bisimilarity

INPUT: A k-PDA $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$ and two configurations $(q,\alpha), (q',\alpha') \in Q \times \mathsf{Stacks}_k(\Gamma)$.

QUESTION: Does $(q,\alpha) \sim (q',\alpha')$ hold in $\mathcal{T}(\mathcal{P})$?

The following proposition is folklore and essentially follows from the fact that every configuration of a k-PDA has only finitely many successors.

▶ Proposition 1. The problem k-PDA-Bisimilarity is in Π_1^0 for each $k \geq 1$.

Post's Correspondence Problem and Variants of it. For two words u, v over some finite alphabet Σ we write $u \preceq v$ if $uw = v$ for some $w \in \Sigma^*$, that is if u is a prefix of v. For a word $w = a_1 \cdots a_n$ with $a_i \in \Sigma$ for each $i \in [1,n]$ we denote its *reverse* by $w^R \stackrel{\text{def}}{=} a_n \cdots a_1$. For a finite (resp. infinite) sequence of finite words $u_1, \ldots, u_n$ (resp. $u_1, u_2, \ldots$) we write $\prod_{i\in[1,n]} u_i \stackrel{\text{def}}{=} u_1 u_2 \cdots u_n$ (resp. $\prod_{i\geq 1} u_i \stackrel{\text{def}}{=} u_1 u_2 \cdots$) to denote their concatentation.

An *instance* of *Post's Correspondence Problem* is given by a tuple $\mathcal{X} = ([1,n], \Sigma, h_1, h_2)$, where $n \in \mathbb{N}$, Σ is a finite *word alphabet*, and where $h_1, h_2 : [1,n]^* \to \Sigma^*$ are homomorphisms. We call $\mathcal{X}$ *increasing* if $|h_1(j)| \leq |h_2(j)|$ for each $j \in [1,n]$. We call $\mathcal{X}$ *non-erasing* if $h_1(j), h_2(j) \neq \varepsilon$ for each $j \in [1,n]$. A *solution* to $\mathcal{X}$ is a mapping $s : [1,m] \to [1,n]$ (equivalently a word $w \in [1,n]^*$), where $m \geq 1$ such that $s(1) = 1$ and moreover $h_1(s(1)\cdots s(m)) =$

$h_2(s(1)\cdots s(m))$. An *ω-solution* to $\mathcal{X}$ is a mapping $s : \mathbb{N} \to [1,n]$ with $s(1) = 1$ such that the following equality over ω-words holds: $\prod_{i\geq 1} h_1(s(i)) = \prod_{i\geq 1} h_2(s(i))$.

▶ Remark. When $\mathcal{X}$ is non-erasing and increasing, the following two statements are equivalent for each $s : \mathbb{N} \to [1,n]$:

- The mapping s is an ω-solution to $\mathcal{X}$.
- $s(1) = 1$ and $h_1(s(1)\cdots s(\ell)) \preceq h_2(s(1)\cdots s(\ell))$ for every $\ell \in \mathbb{N}$.

The classical problem MPCP asks, given an instance $\mathcal{X}$, whether $\mathcal{X}$ has a solution. The infinitary variant ω-MPCP asks, given an instance $\mathcal{X}$, whether $\mathcal{X}$ has an ω-solution.

It was shown in [19] that ω-MPCP is Π^0_1-complete. As already observed in [12], Sipser's Σ^0_1-hardness reduction [22] from the halting problem of Turing machines to MPCP can be transferred to a Π^0_1-hardness reduction to ω-MPCP even when restricting instances to be increasing (by only using Steps 1 to 5 and avoiding Steps 6 and 7 in Section 5.2 of [22]). In fact, by inspecting the homomorphisms constructed by Sipser, one can additionally assume that the instances are non-erasing; the latter was not necessary in the undecidability proofs from [12], but is essential in our hardness proofs. This leads us to the following:

ω-NonErasing-Increasing-MPCP

INPUT: An instance $\mathcal{X} = ([1,n], \Sigma, h_1, h_2)$ that is non-erasing and increasing, i.e. such that $h_1(j), h_2(j) \neq \varepsilon$ and $|h_1(j)| \leq |h_2(j)|$ for each $j \in [1,n]$.

QUESTION: Does $\mathcal{X}$ have an ω-solution?

▶ **Theorem 1** ([22]). *The problem* ω-NonErasing-Increasing-MPCP *is* Π^0_1*-complete.*

3 Undecidability of 2-PDA-Bisimilarity

We prove Π^0_1-hardness of 2-PDA-Bisimilarity by giving a many-to-one reduction from ω-NonErasing-Increasing-MPCP. For this, let us fix an instance $\mathcal{X} = (J, \Sigma, h_1, h_2)$ of ω-NonErasing-Increasing-MPCP. We will construct a 2-PDA $\mathcal{P} = (Q, \mathsf{A}, \Gamma, \Delta)$ and two configurations $(q, [1\bot])$ and $(q', [1\bot])$ such that $\mathcal{X}$ has an ω-solution if and only if $(q, [1\bot]) \sim (q', [1\bot])$ holds in $\mathcal{T}(\mathcal{P})$.

Overview of the Construction.

We start from the pair of configurations $(q, [1\bot])$ (the initial left configuration) and $(q', [1\bot])$ (the initial right configuration), thus both initial configurations consist of just one order-1 stack. We partition the bisimulation game into *three phases*.

Defining $j_1 \stackrel{\text{def}}{=} 1$, in **phase 1** we repeatedly push indices $j_2, j_3, \ldots \in [1,n]$ onto the order-1-stack of *both* configurations and we let Defender choose them by using the technique of "Defender's Forcing". The idea is that Defender's job is to push an infinite sequence of indices that is an ω-solution to $\mathcal{X}$ onto both order-1 stacks ad infinitum. At any situation in the game of the form $(q, [j_\ell \cdots j_1 \bot])$ and $(q', [j_\ell \cdots j_1 \bot])$ Attacker can play the action f to challenge Defender by claiming that $h_1(j_1 \cdots j_\ell)$ is not a prefix of $h_2(j_1 \cdots j_\ell)$.

This leads us to **phase 2** in which Defender wishs to prove $h_1(j_1 \cdots j_\ell) \preceq h_2(j_1 \cdots j_\ell)$. Let $w = j_\ell \cdots j_1 \bot$. We let the game get to the pair of configurations $(t, [w][w][w])$ and $(t', [w][w][w])$. From this position, by again using the "Defender's Forcing" technique and popping on the top-most order-1 stack, we allow Defender to choose a situation of the form $(x, [u^R j_{k-1} \cdots j_1 \bot][w][w])$ and $(x', [u^R j_{k-1} \cdots j_1 \bot][w][w])$, where $1 \leq k \leq \ell$, where u is a prefix of $h_2(j_k)$, and moreover $h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_{k-1})u$.

From the latter pair of configurations, **phase 3** deterministically prints from the left configuration essentially (plus some intermediate symbols) the string $h_1(j_1 \cdots j_\ell)^R$ by first performing a pop_2, and from the right configuration essentially (plus some intermediate symbols) the string $u^R h_2(j_1 \cdots j_{k-1})^R$ by continuing with the current top order-1-stack.

Since we had three copies of w at the end of phase two, we can now perform a pop_2 followed by a single 'wait' on the left configuration, and two pop_2s on the right configuration, so that both then have stack $[w]$. This allows them both to empty their stacks using the same number of pop_1 operations, allowing our 2-PDA to be *normed*. Thus our suggested generalisation of normedness standard for 1-PDA does not help recover decidability.

When describing the rules in detail, we list the rewrite rules of $\mathcal{P}$ in reverse order, i.e. first for **phase 3**, then for **phase 2** and finally for **phase 1**. Adapting the notation from [12], the rewrite rules that are presented in a $\boxed{}$ represent the moves added to implement "Defender's Forcing". These moves allow Defender to render the two configurations equal, and hence trivially bisimilar, if Attacker does not allow Defender to "decide the stack operations".

The Details.

Let $\Gamma \stackrel{\text{def}}{=} [1,n] \cup \Sigma$. The set of states Q, the set of actions A and the transitions Δ of $\mathcal{P}$ are implicitly given by the following rewrite rules. We describe the rules for **phase 3** first.

$x\ \gamma \overset{f}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{pop}_2$	and	$x'\ \gamma \overset{f}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{swap}_\gamma$	for each $\gamma \in \Gamma \cup \{\bot\}$
$y\ a \overset{a}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{pop}_1$	and	$y'\ a \overset{a}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{pop}_1$	for each $a \in \Sigma$
$y\ j \overset{a}{\hookrightarrow}_{\mathcal{P}} y\ \mathsf{swap}_{v^R}$			for each $j \in [1,n]$, where $h_1(j) = va$
		$y'\ j \overset{a}{\hookrightarrow}_{\mathcal{P}} y'\ \mathsf{swap}_{v^R}$	for each $j \in [1,n]$, where $h_2(j) = va$
$y \perp \overset{\bot}{\hookrightarrow}_{\mathcal{P}} z_1\ \mathsf{swap}_\bot$	and	$y' \perp \overset{\bot}{\hookrightarrow}_{\mathcal{P}} z_1'\ \mathsf{pop}_2$	
$z_1 \perp \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_2$	and	$z_1'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_2$	for each $j \in [1,n]$
$z\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} z\ \mathsf{pop}_1$			for each $j \in [1,n]$

For the following lemma, observe that from both the initial configurations in the lemma there is a unique maximal (w.r.t. $\preceq$) word that can be traced.

▶ **Lemma 2.** *Assume $j_1, \cdots, j_\ell \in [1,n]$ with $\ell \geq 1$ and let $0 \leq k \leq \ell$. Assume some 2-stack $\alpha = [u^R j_k \cdots j_1 \bot][j_\ell \cdots j_1 \bot][j_\ell \cdots j_1 \bot]$, where $u \in \Sigma^*$. Then we have*

$$(x,\alpha) \sim (x',\alpha) \quad \textit{if and only if} \quad h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_k)u.$$

Let us add the following rules to Δ in order to implement **phase 2**.

$r_1\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} r_2\ \mathsf{push}_2$	and	$r'_1\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} r'_2\ \mathsf{push}_2$	for each $j \in [1,n]$
$r_2\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} t\ \mathsf{push}_2$	and	$r'_2\ j \overset{f}{\hookrightarrow}_{\mathcal{P}} t'\ \mathsf{push}_2$	for each $j \in [1,n]$
$t \perp \overset{\perp}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_\perp$	and	$t' \perp \overset{\perp}{\hookrightarrow}_{\mathcal{P}} x'\ \mathsf{swap}_\perp$	
$\boxed{t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_\downarrow\ \mathsf{swap}_j}$	and	$t'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_\downarrow\ \mathsf{swap}_j$	for each $j \in [1,n]$
$t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} \overline{t}\ \mathsf{swap}_j$			for each $j \in [1,n]$
$\boxed{t\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_j(w)\ \mathsf{swap}_j}$	and	$t'\ j \overset{p}{\hookrightarrow}_{\mathcal{P}} t'_j(w)\ \mathsf{swap}_j$	for each $j \in [1,n]$ and each prefix w of $h_2(j)$
$\overline{t}\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}}\ t\ \mathsf{pop}_1$	and	$t'_\downarrow\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}}\ t'\ \mathsf{pop}_1$	for each $j \in [1,n]$
		$\boxed{t'_j(w)\ j \overset{\downarrow}{\hookrightarrow}_{\mathcal{P}} t\ \mathsf{pop}_1}$	for each $j \in [1,n]$ and each prefix w of $h_2(j)$
$\overline{t}\ j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{w^R}$	and	$t'_j(w) j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x'\ \mathsf{swap}_{w^R}$	for each $j \in [1,n]$ and each prefix w of $h_2(j)$
	and	$\boxed{t'_\downarrow\ j \overset{\langle j,w\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{w^R}}$	
		$\boxed{t'_j(w)\ j \overset{\langle j,v\rangle}{\hookrightarrow}_{\mathcal{P}} x\ \mathsf{swap}_{v^R}}$	for each $j \in [1,n]$ and all prefixes v, w of $h_2(j)$ s.t. $v \neq w$

▶ **Lemma 3.** *Let $\alpha = j_1 \cdots j_\ell \in [1,n]^\ell$ with $\ell \geq 1$. Then we have*

$$(r_1, [\alpha^R \perp]) \sim (r'_1, [\alpha^R \perp]) \quad \textit{if and only if} \quad h_1(\alpha) \preceq h_2(\alpha).$$

Proof. When inspecting the first two pairs of rules from the previous block, it is clear that $(r_1, [\alpha^R \perp]) \sim (r'_1, [\alpha^R \perp])$ if and only if $(t, [\alpha^R \perp][\alpha^R \perp][\alpha^R \perp]) \sim (t', [\alpha^R \perp][\alpha^R \perp]\alpha^R \perp])$. Define the configuration $\alpha_k \overset{\text{def}}{=} [j_k \cdots j_1 \perp][\alpha^R \perp][\alpha^R \perp]$ for each $0 \leq k \leq \ell$.

"If": Assume $h_1(\alpha) \preceq h_2(\alpha)$. By assumption we have $h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_{k_0 - 1})u$, where $1 \leq k_0 \leq \ell$ and where $u \in \Sigma^*$ is some non-empty prefix of $h_2(j_{k_0})$. First, we have that there exists some bisimulation relation R that relates $(x, [u^R j_{k_0-1} \cdots j_1 \perp][\alpha^R \perp][\alpha^R \perp])$ and $(x', [u^R j_{k_0-1} \cdots j_1 \perp][\alpha^R \perp][\alpha^R \perp])$ by Lemma 2. For each configuration $\zeta \in Q \times \mathsf{Stacks}_2(\Gamma)$, let us define $\mathsf{cl}(\zeta) \overset{\text{def}}{=} \{(\zeta', \zeta') \mid \exists v \in \Sigma^* : \zeta \xrightarrow{v} \zeta'\}$. The reader verifies that the symmetric closure of the following relation is indeed a bisimulation that relates $(t, [\alpha^R \perp][\alpha^R \perp][\alpha^R \perp]) = (t, \alpha_\ell)$ and $(t', [\alpha^R \perp][\alpha^R \perp]\alpha^R \perp]) = (t', \alpha_\ell)$:

$$\begin{array}{llll} & \{(\langle t, \alpha_k\rangle, \langle t', \alpha_k\rangle) \mid k_0 \leq k \leq \ell\} & \cup & \{(\langle \overline{t}, \alpha_k\rangle, \langle t'_\downarrow, \alpha_k\rangle) \mid k_0 < k \leq \ell\} \\ \cup & \bigcup_{k_0 < k \leq \ell, w \preceq h_2(k)} \mathsf{cl}(\langle t'_k(w), \alpha_k\rangle) & \cup & \bigcup_{w \preceq h_2(k_0), w \neq u} \mathsf{cl}(\langle t'_{k_0}(w), \alpha_{k_0}\rangle) \\ \cup & \left\{\left(\langle \overline{t}, \alpha_{k_0}\rangle, \langle t'_{k_0}(u), \alpha_{k_0}\rangle\right)\right\} & & \\ \cup & \mathsf{cl}(\langle t, \alpha_{k_0 - 1}\rangle) & \cup & R \end{array}$$

"Only if": Assume $h_1(j_1 \cdots j_\ell) \not\preceq h_2(j_1 \cdots j_\ell)$. First, recall that we have

$$(x, [u^R j_{k-1} \cdots j_1 \perp][\alpha^R \perp][\alpha^R \perp]) \not\sim (x', [u^R j_{k-1} \cdots j_1 \perp][\alpha^R \perp][\alpha^R \perp]) \tag{1}$$

for every $k \in [1, \ell]$ and each prefix u of $h_2(j_k)$ by Lemma 2.

For proving $(t, [\alpha \perp][\alpha \perp][\alpha \perp]) \not\sim (t', [\alpha \perp][\alpha \perp][\alpha \perp])$ we will show $(t, \alpha_k) \not\sim (t', \alpha_k)$ for each $0 \leq k \leq \ell$ by induction on k.

Induction base. We have to prove $(t, [\bot][\alpha\bot][\alpha\bot]) \not\sim (t', [\bot][\alpha\bot][\alpha\bot])$. When performing action $\bot$ on both configurations we obtain the pair $(x, [\bot][\alpha\bot][\alpha\bot])$ and $(x', [\bot][\alpha\bot][\alpha\bot])$ which is not bisimilar by Lemma 2 since $|\alpha| \geq 1$ and h_2 is non-erasing by assumption.
Induction step. Let $0 \leq k < \ell$. Consider the bisimulation game starting from the pair of configurations (t, α_{k+1}) and (t', α_{k+1}). We will describe a winning strategy for Attacker. Attacker plays $(t, \alpha_{k+1}) \xrightarrow{p} (\bar{t}, \alpha_{k+1})$. We make a case distinction on the answers of Defender. Firstly, assume Defender responds $(t', \alpha_{k+1}) \xrightarrow{p} (t'_{\downarrow}, \alpha_{k+1})$. In the next round, Attacker plays $(t'_{\downarrow}, \alpha_{k+1}) \xrightarrow{\downarrow} (t', \alpha_k)$ and Defender can only respond with $(\bar{t}, \alpha_{k+1}) \xrightarrow{\downarrow} (t, \alpha_k)$. For the resulting pair of configurations it holds $(t, \alpha_k) \not\sim (t', \alpha_k)$ by induction hypothesis. Secondly, assume Defender responds $(t', \alpha_{k+1}) \xrightarrow{p} (t'_{j_{k+1}}(w), \alpha_{k+1})$ for some $w \preceq h_2(j_{k+1})$. Then Attacker can play $(t'_{j_{k+1}}(w), \alpha_{k+1}) \xrightarrow{\langle j_{k+1}, w\rangle} (x', [w^R j_k \cdots j_1][\alpha\bot][\alpha\bot])$ and Defender can only respond with $(\bar{t}, \alpha_{k+1}) \xrightarrow{\langle j_{k+1}, w\rangle} (x, [w^R j_k \cdots j_1][\alpha\bot][\alpha\bot])$ and the resulting pair of configurations is not bisimilar by (1). ◀

Finally, let us add the following rules to Δ for implementing **phase 1**.

$\boxed{q\ k \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} q'_j\ \mathsf{swap}_k}$	and	$q'\ k \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} q'_j\ \mathsf{swap}_k$	for each $j, k \in [1, n]$
$q\ k \overset{\uparrow}{\hookrightarrow}_{\mathcal{P}} \overline{q}\ \mathsf{swap}_k$			for each $k \in [1, n]$
$\overline{q}\ k \overset{j}{\hookrightarrow}_{\mathcal{P}} q\ \mathsf{swap}_{jk}$			for each $k, j \in [1, n]$
		$q'_j\ k \overset{j}{\hookrightarrow}_{\mathcal{P}} q'\ \mathsf{swap}_{jk}$	for each $k, j \in [1, n]$
		$\boxed{q'_j\ k \overset{j'}{\hookrightarrow}_{\mathcal{P}} q\ \mathsf{swap}_{j'k}}$	for each $k, j, j' \in [1, n]$ with $j' \neq j$
$q\ k \overset{f}{\hookrightarrow}_{\mathcal{P}} r_1\ \mathsf{swap}_k$	and	$q'k \overset{f}{\hookrightarrow}_{\mathcal{P}} r'_1\ \mathsf{swap}_k$	for each $k \in [1, n]$

▶ **Lemma 4.** *We have* $(q, [1\bot]) \sim (q', [1\bot])$ *if and only if* $\mathcal{X}$ *has an* ω*-solution.*

One can easily verify that both configurations $(q, [1\bot])$ and $(q', [1\bot])$ are normed. For the following theorem, the lower bound follows from Theorem 1 and and Lemma 4, whereas the upper bound follows from Proposition 1.

▶ **Theorem 5.** *The problem* 2-PDA-Bisimilarity *is* Π^0_1*-complete, even when both input configurations are normed.*

4 The Lower Order Problem

We consider the following class of *lower order* problems for $k \geq 2$ and $k' \in [0, k-1]$ (where a 0-PDA is a finite automaton):

LO$_{k,k'}$

INPUT: A *deterministic* k-PDA $\mathcal{P}$ and a configuration (q, α) of $\mathcal{P}$.
QUESTION: Does there exist a configuration (r, β) of $\mathcal{P}$ with $(q, \alpha) \longrightarrow^* (r, \beta)$ such that $(r, \beta) \sim (r', \beta')$, where (r', β') is a configuration of some k'-PDA $\mathcal{P}'$?

This problem is related to determining whether a program employing order-k functions can ever reach a state from which it could continue using code only constructed with order-k' functions. The case when $k' = 0$ is thus related to the problem of determining whether a (higher-order) recursive program is equivalent to one using only constant memory.

The following holds whether or not $\mathcal{P}'$ is restricted to being deterministic:

▶ **Theorem 6.** $\mathbf{LO}_{k,k'}$ *is undecidable for every* $k \geq 2$ *and* $k' \in [0, k-1]$.

To our knowledge the decidability of $\mathbf{LO}_{1,0}$ and the variant of $\mathbf{LO}_{k,k'}$ for which we require $(r, \beta) = (q, \alpha)$ remain open when $k' \geq 1$ (and $k' \geq 0$ if $\mathcal{P}$ is allowed to be non-deterministic). These stronger problems would be even more pertinent to practical applications.

4.1 Language Recognition

Our construction requires us to use k-PDA as *language recognisers*. In order to do this we extend the model further to: $\mathcal{R} = (Q, q_0, F, \mathsf{A}, \Gamma, \Delta)$ where $q_0 \in Q$ is an *initial control location* and $F \subseteq Q$ is a set of *accepting control locations*. The *language recognised by* $\mathcal{R}$ is:

$$\mathcal{L}(\mathcal{R}) \stackrel{\text{def}}{=} \{\, w \in \mathsf{A}^* \mid (q_0, \bot_k) \xrightarrow{w} (q, \alpha) \text{ for some } q \in F \text{ and } \alpha \in \mathsf{Stacks}(\Gamma) \,\}$$

For simplicity, we allow $\varepsilon \in \mathsf{A}$ and for language recognisers view this as a silent transition. A *reachable* configuration (q, α) is one such that $(q_0, \bot_k) \xrightarrow{w} (q, \alpha)$ for some $w \in \Sigma^*$.

We say that the language $\mathcal{L} \subseteq \Sigma^*$ is *k-complete* if it is recognised by a *deterministic* k-PDA but not by any $(k-1)$-PDA (whether deterministic or non-deterministic). A k-complete language exists for every $k \geq 1$ [6, 7, 16].

In the spirit of [1], we say that a language is *visibly k-complete* for $k \geq 2$ if it is k-complete and we can partition A into $\mathsf{A} = \mathsf{A}^{\uparrow} \uplus \mathsf{A}^{<} \uplus \mathsf{A}^{\downarrow}$ such that it is recognised by a *visible* k-PDA $\mathcal{R}$. A k-PDA $\mathcal{R}$ is *visible* if it has a transition of the form $q\gamma \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{op}$ only if $a \in \mathsf{A}^{\uparrow}$ implies $\mathsf{op} = \mathsf{push}_k$, $a \in \mathsf{A}^{\downarrow}$ implies $\mathsf{op} = \mathsf{pop}_k$ and $a \in \mathsf{A}^{<}$ implies $\mathsf{op} \in \mathsf{Op}_{k-1}$. Thus the language is 'marked' with the order-k pushes and pops performed by some k-PDA recognising it. It is clear that the existence of k-complete languages implies the existence of visibly k-complete languages (pick a k-complete language and some k-PDA recognising it, then convert the k-PDA to recognise a visibly k-complete language by marking the symbols labelling push_k and pop_k transitions—any $(k-1)$-PDA generating the annotated language could also generate the original by removing the annotations).

Let us further say that a deterministic k-PDA $\mathcal{R}$ is *total* if $\mathcal{R}$ is real-time (i.e. without ε-transitions) and for every reachable configuration (q, α) and every $a \in \mathsf{A}$, there exists a transition of the form $q\ \mathsf{top}_1(\alpha) \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{op}$ such that $\mathsf{op}(\alpha)$ is defined.

We say that a deterministic visible k-PDA $\mathcal{R}$ recognising a visibly k-complete language is *visibly-total* if $\mathcal{R}$ is real-time and for every reachable configuration (q, α) and every $a \in \mathsf{A}^{<}$ there exists a transition of the form $q\ \mathsf{top}_1(\alpha) \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{op}$ such that $\mathsf{op}(\alpha)$ is defined, for every $a \in \mathsf{A}^{\uparrow}$ a transition $q\ \mathsf{top}_1(\alpha) \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{push}_k$ and for every $a \in \mathsf{A}^{\downarrow}$ there is a transition of the form $q\ \mathsf{top}_1(\alpha) \overset{a}{\hookrightarrow}_{\mathcal{R}} q'\mathsf{pop}_k$ (whether or not $\mathsf{pop}_k(\alpha)$ is defined).

▶ **Lemma 7.** *For every* $k \geq 1$ *there exists a (visibly)* k*-complete language that is recognised by a deterministic (visibly)-total* k*-PDA.*

4.2 Outline of the undecidability of $\mathbf{LO}_{k,k'}$ with $k \geq 2$

In all cases we work by a reduction from (the finitary variant of) MPCP. We fix an MPCP instance $\mathcal{X} = ([1, n], \Sigma, h_1, h_2)$ and for each $\mathbf{LO}_{k,k'}$ we construct a k-PDA $\mathcal{P}_{\mathcal{X}}$ such that there is a solution to the lower order problem if and only if there is a solution to $\mathcal{X}$. We have two separate constructions, one for the case $k - k' \geq 2$ and one for the case $k' = k - 1$. First, let us assume $k - k' \geq 2$. The idea is that for each potential solution w to $\mathcal{X}$ there is a configuration c_w that is reachable from a configuration $(q_0, \bot_k)$ whose k-stack is of the form $[\alpha][\mathsf{swap}_w(\bot_{k-1})]$ for some $(k-1)$-stack α. From c_w the deterministic k-PDA $\mathcal{P}_{\mathcal{X}}$ can

simulate on its stack α some deterministic $(k-1)$-PDA recognizing some $(k-1)$-complete language. However, at any point $\mathcal{P}_\mathcal{X}$ threatens to pop $[\alpha]$ and then deterministically output $h_1(w)^R$ in case the current control location of the simulation is in some accepting control location or else to deterministically output $h_2(w)^R$ in case the current control location of the simualation is in some non-accepting control location. In case $h_1(w) = h_2(w)$ the distinction between accepting and rejecting control locations cannot be made by observing this behaviour and so there will be a finite automaton bisimilar to $\mathcal{P}_\mathcal{X}$.

Let us now assume $k' = k-1$. We follow a similar idea, but this time simulate an automaton for a k-complete language. However this simulation may lead to k-stacks of the form $[\alpha_m]\cdots[\alpha_1][\mathsf{swap}_w(\bot_{k-1})]$ for arbitrarily large m. This means that to carry out the threat of printing $h_1(w)^R$ or $h_2(w)^R$ it is necessary to make an unbounded number of pop_k operations. If w is a solution to $\mathcal{X}$ so that $h_1(w) = h_2(w)$ then a bisimilar $\mathcal{P}'$ will need to 'know' m in order to synchronise with this preparation to print $h_1(w)$ or $h_2(w)$. It thus needs to be equipped with a counter, but since $k-1 \geq 1$, this is possible. Concerning the proof, there is a simple modification exploiting visibly total automata recognising k-complete visible languages, thus we have decidided to relegate this case to the long version.

4.3 The undecidability of $\mathsf{LO}_{k,k'}$ for $k \geq 2$ and $k-2 \geq k' \geq 0$.

We fix a deterministic $(k-1)$-PDA $\mathcal{R} = (Q_\mathcal{R}, q_{0\mathcal{R}}, F_\mathcal{R}, \mathsf{A}_\mathcal{R}, \Gamma_\mathcal{R}, \Delta_\mathcal{R})$ that recognises a $(k-1)$-complete language. Due to Lemma 7 we may assume that it is *total-deterministic*. We do not require any visibility assumption in this section. We construct a k-PDA $\mathcal{P}_\mathcal{X} = (Q, \mathsf{A}, \Gamma, \Delta)$ and take a configuration $(q_0, \bot_k)$ as input to the problem. We will progressively introduce Δ (and thereby Q, A, and Γ).

First we have rules responsible for 'guessing' a solution to MPCP and spawning $\mathcal{R}$:

$$q_0 \bot \overset{\#_1}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0\ \mathsf{swap}_{1\bot} \quad \text{and} \quad q_0\ j \overset{\#_{j'}}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0\ \mathsf{swap}_{j'j} \quad \text{for each } j, j' \in [1,n]$$
$$q_0\ j \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0'\ \mathsf{swap}_j \quad \text{and} \quad q_0'\ j \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0''\ \mathsf{push}_k \quad \text{for each } j \in [1,n]$$
$$q_0''\ j \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0''\ \mathsf{pop}_1 \quad \text{and} \quad q_0''\ \bot \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_{0\mathcal{R}}\ \mathsf{swap}_\bot \quad \text{for each } j \in [1,n]$$

We then add every rule in $\Delta_\mathcal{R}$ to Δ. Finally we have rules responsible for printing out the image of the alleged solution under either h_1 or h_2:

$$q_f\ \gamma \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_1^-\ \mathsf{pop}_k \quad \text{and} \quad q\ \gamma \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_2^-\ \mathsf{pop}_k \quad \text{for each } \gamma \in \Gamma_\mathcal{R},\ q_f \in F_\mathcal{R},\ q \in Q_\mathcal{R} - F_\mathcal{R}$$
$$y_1^-\ j \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_1\ \mathsf{push}_k \quad \text{and} \quad y_2^-\ j \overset{\#}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_2\ \mathsf{push}_k \quad \text{for each } j \in [1,n]$$
$$y_1\ a \overset{a}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_1\ \mathsf{pop}_1 \quad \text{and} \quad y_2\ a \overset{a}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_2\ \mathsf{pop}_1 \quad \text{for each } a \in \Sigma$$
$$y_1\ j \overset{a}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_1\ \mathsf{swap}_{v_1^R} \quad \text{and} \quad y_2\ j \overset{a}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} y_2\ \mathsf{swap}_{v_2^R} \quad \text{f. e. } j \in [1,n]\text{: } h_1(j) = v_1 a,\ h_2(j) = v_2 a$$
$$y_1 \bot \overset{\mathsf{reset}}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0'\ \mathsf{pop}_k \quad \text{and} \quad y_2 \bot \overset{\mathsf{reset}}{\hookrightarrow}_{\mathcal{P}_\mathcal{X}} q_0'\ \mathsf{pop}_k$$

We have borrowed the rules for y_1 and y_2 from the previous construction, and so from the proof of Lemma 2 we get:

▶ **Lemma 8.** *Suppose that we have a stack* $\alpha = [\mathsf{swap}_{j_\ell \cdots j_1 \bot}(\bot_{k-1})]\ [\mathsf{swap}_{j_\ell \cdots j_1 \bot}(\bot_{k-1})]$ *where* $j_1, \ldots, j_\ell \in [1,n]$. *Then we have* $(y_i, \alpha) \xrightarrow{u\mathsf{reset}} (q_0', \mathsf{pop}_k(\alpha))$ *if and only if* $u = h_i(j_1 \cdots j_\ell)^R$, *for each* $i \in \{1,2\}$.

We have the following characterisation of reachable configurations of $\mathcal{P}_\mathcal{X}$:

▶ **Lemma 9.** *Let (p,α) be a configuration of $\mathcal{P}_{\mathcal{X}}$ with $(q_0,\perp_k) \longrightarrow^* (p,\alpha)$. Then one of the following holds:*

- $p \in \{q_0, q_0', y_1^-, y_2^-\}$ *and* $\alpha = [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *for some* $j_1,\ldots,j_\ell \in [1,n]$ *with* $j_1 = 1$.
- $p = q_0''$ *and* $\alpha = [\mathsf{swap}_{j_{\ell'} \cdots j_1 \perp}(\perp_{k-1})]\ [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *for some* $j_1,\ldots,j_\ell \in [1,n]$ *with* $j_1 = 1$ *and* $0 \le \ell' \le \ell$.
- $p \in \{y_1, y_2\}$ *and* $\alpha = [\mathsf{swap}_{w j_{\ell'} \cdots j_1 \perp}(\perp_{k-1})]\ [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *for some* $j_1,\ldots,j_\ell \in [1,n]$ *with* $j_1 = 1$ *and* $0 \le \ell' \le \ell$ *and* $w \in \Sigma^*$.
- $p \in Q_{\mathcal{R}}$ *and* $\alpha = [\beta]\ [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})]$ *for some* $j_1,\ldots,j_\ell \in [1,n]$ *with* $j_1 = 1$ *and* $0 \le \ell' \le \ell$ *where* (p,β) *is a reachable configuration of* $\mathcal{R}$.

Proof. By induction on the length of a run from $(q_0,\perp_k)$ witnessing reachability. ◀

Thanks to the **reset** transitions we have the following:

▶ **Lemma 10.** *Assume $(q_0,\perp_k) \longrightarrow^* c$ for some configuration c of $\mathcal{P}$ such that $c \sim c'$ for some configuration c' of some k'-PDA with $k - k' \ge 2$. Then $(q_0,\perp_k) \longrightarrow^* c_0$ where $c_0 = (q_0', [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})])$, for some $j_1,\ldots,j_\ell \in [1,n]$ and $j_1 = 1$. Moreover, there is some configuration c_0' of $\mathcal{P}'$ with $c' \longrightarrow^* c_0'$ and $c_0 \sim c_0'$.*

Proof. Since $c = (p,\alpha)$ must satisfy one of the conditions Lemma 9, by considering each case in turn it can easily be verified that $c \xrightarrow{u} c_0$ for c_0 of the requisite form with $u \in \mathsf{A}^*$. Since $c \sim c'$ we must have $c' \xrightarrow{u} c_0'$ with $c_0 \sim c_0'$ for some configuration c_0' of $\mathcal{P}'$. ◀

Let us first consider the case when $\mathcal{X}$ has no solution. Due to Lemma 10 it is sufficient to consider $\mathcal{P}$-configurations c of the form (q_0',α). Since the image of the sequence of indices in α under h_1 and h_2 must differ, it can be shown that any automaton with configuration bisimilar to c could be converted to one recognising $\mathcal{L}(\mathcal{R})$. Completeness then gives:

▶ **Lemma 11.** *If $\mathcal{X}$ has no solution, then there is no reachable configuration c of $\mathcal{P}$ and k'-PDA $\mathcal{P}'$ with configuration c' such that $c \sim c'$.*

Now we prove the converse.

▶ **Lemma 12.** *If $\mathcal{X}$ has a solution, then $(q_0,\perp_k) \longrightarrow^* c$ for some configuration c of $\mathcal{P}$ and a deterministic real-time k'-PDA $\mathcal{P}'$ with configuration c' such that $c \sim c'$.*

Proof. Let $j_1 \cdots j_\ell$ be a solution to $\mathcal{X}$. Let $c \stackrel{\text{def}}{=} (q_0', [\mathsf{swap}_{j_\ell \cdots j_1 \perp}(\perp_{k-1})])$ with $h_1(j_1 \cdots j_\ell) = h_2(j_1 \cdots j_\ell) = a_1 \cdots a_m$ with $a_i \in \Sigma$ for each $i \in [1,m]$. We then take $\mathcal{F} = (S, \mathsf{A}, \{\xrightarrow{a}_{\mathcal{F}} | a \in \mathsf{A}\})$ to be the following deterministic finite transition system, which in particular is the transition system of a k'-PDA and set $c' \stackrel{\text{def}}{=} s$. We set $S \stackrel{\text{def}}{=} \{s,t,u\} \cup \{x_i \mid 0 \le i \le \ell\} \cup \{y_i \mid 0 \le i \le m\}$ and define and $\xrightarrow{a}_{\mathcal{F}}$ for each $a \in \mathsf{A}$ as follows:

$s \xrightarrow{\#}_{\mathcal{F}} x_\ell$	and	$x_\ell \xrightarrow{\#}_{\mathcal{F}} x_{i-1}$	for each $i \in [1,\ell]$
$x_0 \xrightarrow{\#}_{\mathcal{F}} t$	and	$t \xrightarrow{a}_{\mathcal{F}} t$	for each $a \in \mathsf{A}_{\mathcal{R}}$
$t \xrightarrow{\#}_{\mathcal{F}} u$	and	$u \xrightarrow{\#}_{\mathcal{F}} y_m^-$	
$y_0^- \xrightarrow{\mathsf{reset}} s$	and	$y_i^- \xrightarrow{a_i}_{\mathcal{F}} y_{i-1}^-$	for each $1 \le i \le m$

Recalling that $\mathcal{R}$ is total-deterministic and real-time and so from every configuration may make an a-transition for any $a \in \Sigma_{\mathcal{R}}$, the reader can verify that the symmetric closure

of the following relation is a bisimulation:

$$\begin{aligned}
&\{(c,s)\} \cup \{((q_{\mathcal{R}},\beta),t) \mid c \longrightarrow^* (q_{\mathcal{R}},\beta) \text{ and } q_{\mathcal{R}} \in Q_{\mathcal{R}}\}\\
\cup\ &\{((y_j^-,\alpha),u) \mid c \longrightarrow^* (y_j^-,\alpha)\}\\
\cup\ &\{((q_0'',\alpha_{\ell'}),x_{\ell'}) \mid \alpha_{\ell'} = [\mathsf{swap}_{j_{\ell'}\cdots j_1\perp}(\perp_{k-1})]\ [\mathsf{swap}_{j_\ell\cdots j_1\perp}(\perp_{k-1})]) \quad 0 \le \ell' \le \ell\}\\
\cup\ &\{((y_j,\alpha_j^i),\bar{y_i}) \mid (y_j,[\mathsf{swap}_{j_\ell\cdots j_1\perp}(\perp_{k-1})]\ [\mathsf{swap}_{j_\ell\cdots j_1\perp}(\perp_{k-1})]) \xrightarrow{a_m\cdots a_{i+1}} (y_j,\alpha_j^i),\\
&\quad 1 \le j \le 2,\ 0 \le i \le m\}
\end{aligned}$$

◀

5 Further Directions

We believe some limited generalisation of undecidability for $\mathbf{LO}_{k,k'}$ to the case when $\mathcal{P}'$ may also range over *collapsible pushdown automata* [8] is possible. We expect it to be possible to adapt the construction to use the recent hierarchy theorem by Kartzow and Parys for deterministic CPDA [13]. Indeed in the light of [18] one might expect to get a version where $\mathcal{P}$ is a 2-CPDA and $\mathcal{P}'$ can range over deterministic PDA of *any order*. One obstacle is when $(k-k')=1$ and $\mathcal{P}'$ needs to be able to keep track of the height of the k-stack of $\mathcal{P}$, meaning that a simple extension to the CPDA case would require one to prohibit 'k-links'.

References

1 Rajeev Alur and P Madhusudan. Visibly Pushdown Languages. In *STOC*, pages 202–211. ACM, 2004.

2 J. L. Balcázar, J. Gabarró, and M. Santha. Deciding Bisimilarity is P-Complete. *Formal Asp. Comput.*, 4(6A):638–648, 1992.

3 S. Böhm, S. Göller, and P. Jančar. Bisimilarity of one-counter processes is PSPACE-complete. In *Proc. of CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 2010.

4 T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *Proceedings of ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569. Springer, 2003.

5 A. Carayol and S. Wöhrle. The Caucal Hierarchy of Infinite Graphs in Terms of Logic and Higher-Order Pushdown Automata. In *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

6 W. Damm. An algebraic extension of the Chomsky Hierarchy. In *MFCS*, 1979.

7 W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71(1-2):1–32, October 1986.

8 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible Pushdown Automata and Recursion Schemes. In *LICS*. IEEE Computer Society, 2008.

9 D. Janin and I. Walukiewicz. On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In *Proc. of CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

10 P. Jančar. Strong Bisimilarity on Basic Parallel Processes is PSPACE-complete. In *Proc. of LICS*, pages 218–227. IEEE Computer Society, 2003.

11 P. Jančar. Decidability of dpda language equivalence via first-order grammars. In LICS, 2012.

12 P. Jančar and Jirí Srba. Undecidability of bisimilarity by defender's forcing. *J. ACM*, 55(1), 2008.

13 A. Kartzow and P. Parys. Strictness of the Collapsible Pushdown Hierarchy. In *MFCS*, 2012.

14 T.r Knapik, D. Niwinski, and P. Urzyczyn. Higher-Order Pushdown Trees Are Easy. In *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

15 A. Kucera. Efficient Verification Algorithms for One-Counter Processes. In *ICALP*, pages 317–328, 2000.

16 A N Maslov. Multilevel Stack Automata. *Problems of Information Transmission*, (12):38–43, 1976.

17 F. Moller and A. M. Rabinovich. Counting on CTL^*: on the expressive power of monadic path logic. *Inf. Comput.*, 184(1):147–159, 2003.

18 P. Parys. On the Significance of the Collapse Operation. In *LICS*, volume 1, 2012.

19 K. Ruohonen. Reversible machines and post's correspondence problem for biprefix morphisms. *Elektronische Informationsver- arbeitung und Kybernetik*, 21(12):579–595, 1985.

20 G. Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001.

21 G. Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM J. Comput.*, 34(5):1025–1106, 2005.

22 M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

23 J. Srba. *Roadmap of Infinite results*, volume Vol 2: Formal Models and Semantics. World Scientific Publishing Co., 2004. http://www.brics.dk/~srba/roadmap.

24 R.E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967.

25 C. Stirling. Deciding DPDA Equivalence Is Primitive Recursive. In *Proc. of ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 821–832. Springer, 2002.

26 C. Stirling. Second-order simple grammars. In *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 2006.

27 Leslie G. Valiant. Regularity and related problems for deterministic pushdown automata. *J. ACM*, 22(1):1–10, January 1975.

28 J. van Benthem. *Modal Correspondence Theory*. PhD thesis, University of Amsterdam, 1976.

29 R. J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

Scope-bounded Multistack Pushdown Systems: Fixed-Point, Sequentialization, and Tree-Width*

Salvatore La Torre[1] **and Gennaro Parlato**[2]

1 **Dipartimento di Informatica,**
Università degli Studi di Salerno, Italy
slatorre@unisa.it
2 **School of Electronics and Computer Science**
University of Southampton, UK
gennaro@ecs.soton.ac.uk

Abstract

We present a novel fixed-point algorithm to solve reachability of multi-stack pushdown systems restricted to runs where matching push and pop transitions happen within a bounded number of context switches. The followed approach is compositional, in the sense that the runs of the system are summarized by bounded-size interfaces. Moreover, it is suitable for a direct implementation and can be exploited to prove two new results. We give a sequentialization for this class of systems, i.e., for each such multi-stack pushdown system we construct an equivalent single-stack pushdown system that faithfully simulates the behavior of each thread. We prove that the behavior graphs (multiply nested words) for these systems have bounded tree-width, and thus a number of decidability results can be derived from Courcelle's theorem.

1998 ACM Subject Classification D.2.4 Software/Program Verification.

Keywords and phrases Model-checking, multi-threaded programs, sequentialization, tree-width.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.173

1 Introduction

Multi-stack pushdown systems (MPDS) accurately capture the control-flow of concurrent programs communicating via shared memory, and thus, are widely used as an abstract model of such programs in several analysis problems, such as reachability and more in general model-checking.

It is well known that MPDS with two stacks can simulate Turing machines. A recent line of research has concerned with decidable syntactic restrictions that limit the behaviors of the general model, such as bounding the number of context-switches [21] or the number of phases [9, 10] in a run.

Last year in [15], a new restriction that limits the *scope* of matching push and pop transitions in terms of number of context switches (*scope-bounded restriction*) has been considered. With this limitation, the analysis is carried over only the system executions where each symbol pushed onto a stack is either popped within a bounded number of context switches or is never popped. As a matter of fact, the scope-bounded restriction is strictly more permissive than bounded-context switching, in fact it allows us to account for computations with unboundedly many contexts, and thus, with an unbounded number of interactions between the threads (see [15]).

* This work was partially funded by the FARB grants 2010-2012 Università degli Studi di Salerno (Italy).

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 173–184

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

Also, in [15] the reachability of MPDS under the scope-bounded restriction is shown to be PSPACE-complete. For an n stack MPDS, the given decision algorithm characterizes the configurations which are k-scoped reachable by computing tuples which store: (1) the reached control state, (2) the (top) portion of the stack contents of all the symbols that have been pushed in the last k rounds, and (3) the control states at the context-switches in the last round (computations are arranged in rounds of thread executions where each thread is activated exactly once). The stack contents are summarized as finite automata of a fixed form: each automaton has as states k copies of the control states along with an initial state, and differ from each other only in the transitions.

In this paper, we adopt the same restriction on the MPDS computations and contribute to this research in several ways.

As a first result, we develop a new algorithm to solve the scope-bounded reachability problem for MPDS. The solution we propose is fixed-point and uses the concept of interface of a thread computation introduced in [13]. A *thread-interface* simply summarizes the executions of a thread in consecutive rounds of executions of a system computation, by storing the control states of the starting and ending configurations in each round. An interesting feature of thread-interfaces is that they can be used compositionally to summarize entire runs. A key result that we prove here and exploit to design our fixed-point algorithm is that it is sufficient to store n-tuples of fragments of thread-interfaces over at most k rounds, to reconstruct the summaries of entire k-scoped runs of an MPDS with n stacks.

The proposed algorithm have a simpler formulation than that given in [15] and seems to be more suitable for efficient implementation. Thread interfaces are a simpler artifact than finite automata and can be easily encoded for efficient symbolic search. In fact, our fixed-point algorithm has a direct implementation in the tool GETAFIX, a framework that supports the writing in a fixed-point calculus of model-checkers for sequential and concurrent Boolean programs (see [11]). Moreover, dealing with simpler tuples seems to avoid some redundancy. In fact, if on the one side computing such automata is computationally equivalent to compute thread interfaces, on the other side our algorithm searches over essentially Q^{2kn} tuples while the previous one over $Q^n 2^{n(k^2Q^2+\mu)}$ tuples, where Q denotes the set of control states and μ denotes subquadratic terms in k and Q.

The approach followed in our above fixed point algorithm can be used in two directions to obtain interesting results which constitute the other contributions of this paper.

First, the fixed-point rules used to compose, accumulate and simplify the thread-interfaces in our algorithm can be re-used to construct a single-stack pushdown system that simulates the k-scoped runs of an n-stack MPDS. For computer programs, this corresponds to a *sequentialization*, i.e., a transformation of a concurrent program into an *equivalent* sequential one. Sequentializations have recently received great attention in the context of program verification with the goal of performing the analysis of concurrent programs via tools designed for sequential ones (e.g., see [18, 5]). Several tools have been developed on this paradigm and have allowed to find bugs that could not be found with other approaches [8], or even prove programs entirely correct [13].

Second, we show that multiply nested words, which allow us to represent runs of MPDS with graphs, have a bounded tree-width when restricting to bounded scope. Again, the executions of the fixed point algorithm are the key concept of this proof. Moreover, since this class of multiply nested words can be captured in the MSO logic, we can inherit all the decidability results of [20]. In particular, all properties that can be expressed in MSO can be shown decidable using this result, including the decidability of linear temporal logic.

Related work. Besides the already cited research there are a few other works which

are related to ours. We start mentioning some recent results that have concerned MPDS restricted to scope-bounded computations that have followed the results presented in this paper. In [7], the authors show that bounded-scoped multiply nested words have bounded tree-width using the notion of split-width there introduced. In [1], the model-checking of scope-bounded MPDS is shown to be EXPTIME-complete for linear-time temporal logic (LTL). Independently, in [16], a logic for multiply nested words which extends LTL and allows to capture the call-return relations within each tread (MultiCaRet) is introduced and, among other results, the related model-checking and satisfiability problems are shown to be EXPTIME-complete when restricting to scope-bounded computations.

The notion of bounded-context switching has been successfully used for: model-checking tools for concurrent Boolean programs [11, 18, 22] and Boolean abstractions of parameterized programs [13]; sequentializations of shared-memory concurrent programs [12, 18] and their use with SMT solvers to find errors in concurrent programs [8];sequentialization of multiprocessor programs communicating through asynchronous message passing [4]; sequentializations of shared-memory parameterized programs [14]; translation of concurrent programs under total store ordering memory model to concurrent programs under sequential consistency memory model [2]; model-checking of programs with dynamic thread creation [3]; analysis of systems with heaps [6], and weighted pushdown systems [19].

2 Multistack Pushdown Systems

Given two positive integers i and j, $i \leq j$, we denote with $[i, j]$ the set of integers k with $i \leq k \leq j$, and with $[j]$ the set $[1, j]$.

A multi-stack pushdown system consists of a finite number of pushdown automata each of which with its local stack, that communicate through the shared control states. Multi-stack pushdown system is a faithful model to represent concurrent Boolean programs, where each pushdown component models a single thread and the shared control states can be used to allow shared communication among them.

▶ **Definition 1.** (MULTI-STACK PUSHDOWN SYSTEMS) Let $n \in \mathbb{N}$. A *n-stack pushdown system* (n-MPDS) is a tuple $M = (Q, q_0, \Gamma, \{(\delta_i^{int}, \delta_i^{push}, \delta_i^{pop})\}_{i\in[n]})$ where Q is a finite set of control states, $q_0 \in Q$ is the initial state, Γ is a finite stack alphabet, and for every $i \in [n]$, $\delta_i^{int} \subseteq (Q \times Q)$ is a set of internal transitions and $\delta_i^{push}, \delta_i^{pop} \subseteq (Q \times \Gamma \times Q)$ are respectively push and pop transitions involving the i'th stack. For every $i \in [n]$, with M_i we denote the i'th thread of M, i.e., the 1-MPDS $(Q, q_0, \Gamma, \{(\delta_i^{int}, \delta_i^{push}, \delta_i^{pop})\})$. ◀

An M *configuration* is a tuple $C = \langle q, \{w_i\}_{i\in[n]}\rangle$, where $q \in Q$ and each $w_i \in \Gamma^*$ is the content of the i'th stack. Moreover, C is *initial* if $q = q_0$ and $w_i = \varepsilon$, for every $i \in [n]$. Let $Act_i = \{int_i, push_i, pop_i\}$ be the set of *actions* of thread M_i, and $Act = \bigcup_{i\in[n]} Act_i$ be the set of all *actions* of M. A transition between two configurations over an action $\sigma \in Act$ is defined as follows:
$\langle q, \{w_i\}_{i\in[n]}\rangle \xrightarrow{\sigma}_M \langle q', \{w_i'\}_{i\in[n]}\rangle$ if one of the following holds for some $i \in [n]$

[Internal] $\sigma = int_i$, $(q, q') \in \delta_i^{int}$, and $w_h' = w_h$ for every $h \in [n]$.
[Push] $\sigma = push_i$, $(q, \gamma, q') \in \delta_{push}^i$, $w_i' = \gamma.w_i$, and $w_h' = w_h$ for $h \in ([n] \setminus \{i\})$.
[Pop] $\sigma = pop_i$, $(q, \gamma, q') \in \delta_{pop}^i$, $w_i = \gamma.w_i'$, and $w_h' = w_h$ for $h \in ([n] \setminus \{i\})$.

A *run* ρ of M from C_0 to C_m, with $m \geq 0$, denoted $C_0 \rightsquigarrow_M C_m$, is a possibly empty sequence of transitions $C_{j-1} \xrightarrow{\sigma_j}_M C_j$, for every $j \in [m]$. Furthermore, ρ is a *computation* of M if C_0 is initial.

Scope-bounded Runs. Let $\rho = C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} \dots C_{m-1} \xrightarrow{\sigma_m}_M C_m$ be a run of an n-MPDS M. We associate to each transition in ρ a *round number*. The map $round^\rho : [m] \to \mathbb{N}$ is inductively defined as follows:

$$round^\rho(r) = \begin{cases} 1 & \text{if } r = 1; \\ round^\rho(r-1) + 1 & \text{if } (r > 1) \ \&\ (\sigma_{r-1} \in Act_i) \\ & \quad \&\ (\sigma_r \in Act_{i'}) \ \&\ (i > i'); \\ round^\rho(r-1) & \text{otherwise.} \end{cases}$$

For any M run ρ and $i \in [n]$, $\mu_i^\rho(s,t)$ is the predicate that holds true whenever the t'th transition of ρ pops the symbol pushed on stack i at the s'th transition. Formally, μ_i^ρ is the unique predicate over the set $[m]^2$ such that: if $\mu_i^\rho(s,t)$ holds true then $s < t$, $\sigma_s = push_i$, $\sigma_t = pop_i$, and

- for every $t' \in [m]$ with $s < t' < t$, if $\sigma_{t'} = pop_i$ then there is an index $s' \in [m]$ with $s < s' < t'$ such that $\sigma_{s'} = push_i$ and $\mu_i^\rho(s',t')$ holds true;
- for every $s' \in [m]$ with $s < s' < t$, if $\sigma_{s'} = push_i$ then there is an index $t' \in [m]$ with $s' < t' < t$ such that $\sigma_{t'} = pop_i$ and $\mu_i^\rho(s',t')$ holds true.

Furthermore, if $\sigma_t = pop_i$ then there is an s such that $\mu_i^\rho(s,t)$ holds true.

For $k \in \mathbb{N}$, ρ is *k-scoped* iff for every $i \in [n]$ and two indices $s,t \in [m]$ if $\mu_i^\rho(s,t)$ holds true then $(round^\rho(t) - round^\rho(s)) < k$. In other words, in any k-scoped run any pop operation retrieves from the stack a symbol that has been pushed within the last k rounds.

To illustrate the above concepts consider the run of a 2-MPDS in Figure 1. To simplify the representation, we have omitted the stack contents and reported only the control state of the configurations. Also, we have repeated the control location ending a row at the beginning of the following one such that each row corresponds to a whole round. The states where the control switches from the first to the second thread in each round have been aligned under a column. To emphasize the matching of push and pop we use a different alphabet letter for denoting each push and the same letter with a bar for the matching pop. We use subscripts to distinguish among the different threads.

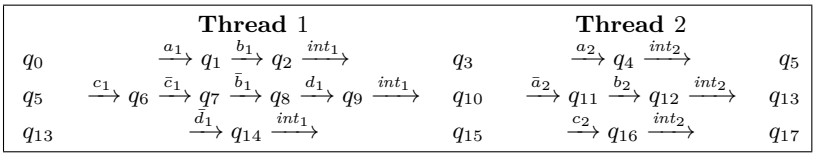


Figure 1 A sample 3-round run of a 2-MPDS.

The run of Figure 1 is k-scoped for any $k \geq 2$ since matching push/pop spans over at most 2 rounds, and it is not 1-scoped, in fact for example the push denoted b_1 in round 1 is matched in round 2.

3 Interfaces

In this section, we introduce the concept of *thread-interface* which is central for the paper. Informally, a thread-interface summarizes for a single thread the computation within consecutive rounds, and under some conditions, can be composed with the thread-interfaces of the other threads to summarize entire runs of an MPDS. We show that when restricting to k-scoped runs, the whole computation of a single thread across unboundedly many rounds can be indeed captured by composing thread-interfaces over at most k rounds. This will be exploited in the next section to give a fixed-point algorithm to solve the reachability problem restricted to k-scoped runs of an MPDS.

▶ **Definition 2.** (THREAD-INTERFACE) Let $M = (Q, q_0, \Gamma, \{(\delta_i^{int}, \delta_i^{push}, \delta_i^{pop})\}_{i\in[n]})$ be an n-MPDS. For each $i \in [n]$, an *i-thread-interface* of M is a possibly empty tuple $I = \langle in_j, out_j\rangle_{j\in[r]}$, for some $r \in \mathbb{N}$ (the *dimension* of I, also denoted $\dim(I)$), such that if $r > 0$ there exist r runs $\pi_1, \pi_2, \ldots, \pi_r$ of M_i in which

- for every $j \in [r]$, $\pi_j = \langle in_j, w_j\rangle \rightsquigarrow_{M_i} \langle out_j, w'_j\rangle$ is a run of M_i;
- $w_1 = \epsilon$, and for every $j \in [r-1]$, $w_{j+1} = w'_j$. ◀

Fix for the rest of the section a 2-MPDS P with a run as in Figure 1. From the above definition the tuple $T_1 = (\langle q_0, q_3\rangle, \langle q_5, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$ is a 1-thread-interface of P of dimension 3 and $T_2 = (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle, \langle q_{15}, q_{17}\rangle)$ is a 2-thread-interface of P of dimension 3. Note that since a run has possibly zero transitions, also $T_3 = (\langle q_0, q_3\rangle, \langle q_5, q_5\rangle, \langle q_5, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$ and $T_4 = (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle, \langle q_{15}, q_{15}\rangle)$ are thread-interfaces.

For $h = 1, 2$, let $I_h = \langle in_j^h, out_j^h\rangle_{j\in[r_h]}$ be an i-thread-interface of M, for some $i \in [n]$. We define two internal operations over thread-interfaces of a given thread. With $I_1 \bowtie_1 I_2$ we denote the tuple obtained by appending I_2 to I_1. Formally, $I_1 \bowtie_1 I_2 = \langle in_j, out_j\rangle_{j\in[r_1+r_2]}$ where $in_j = in_j^1$ and $out_j = out_j^1$ for $j \in [r_1]$, and $in_{r_1+j} = in_j^2$ and $out_{r_1+j} = out_j^2$ for $j \in [r_2]$. The other operation is a variation of $\bowtie_1$ where the last pair of I_1 is composed with the first pair of I_2. It is defined when I_1 and I_2 are both not empty. Formally, if $r_1, r_2 > 0$ and $out_{r_1}^1 = in_1^2$, then we denote with $I_1 \bowtie_2 I_2$ the tuple $\langle in_j, out_j\rangle_{j\in[r_1+r_2-1]}$ where $in_j = in_j^1$ and $out_j = out_j^1$ for $j \in [r_1 - 1]$, $in_{r_1} = in_{r_1}^1$, $out_{r_1} = out_1^2$, and $in_{r_1+j} = in_{j+1}^2$ and $out_{r_1+j} = out_{j+1}^2$ for $j \in [r_2 - 1]$.

Directly from the definition of thread-interface we get that both compositions define thread interfaces.

▶ **Lemma 3.** *Let $I_h = \langle in_j^h, out_j^h\rangle_{j\in[r_h]}$ be a i-thread-interface of M, for some $i \in [n]$ and $h = 1, 2$.*

$I_1 \bowtie_1 I_2$ is a i-thread-interface of dimension $r_1 + r_2$.

If $out_{r_1}^1 = in_1^2$, then $I_1 \bowtie_2 I_2$ is a i-thread-interface of dimension $r_1 + r_2 - 1$.

The two compositions $\bowtie_1$ and $\bowtie_2$ are sufficient to fully characterize, by thread-interfaces of bounded dimension all the thread-interfaces "canonically" defined by scope-bounded runs of an MPDS. Given an m-round run ρ of an n-MPDS M, a i-thread-interface $I = \langle in_j, out_j\rangle_{j\in[m]}$ is *canonical for ρ* if along ρ for each round j the computation of thread M_i starts at in_j and ends at out_j. The idea is thus to capture with each i-thread-interface a portion ρ' of the run, where all the occurrences of pushes over the i'th stack are either matched within ρ' or are never matched in the whole run. Due to the k-scoped restriction, for all matched pushes the matching pop transition must occur within k rounds, and since the matching pairs of push/pops define a nested relation, each such portion ρ' can be taken such that it spans over at most k rounds.

As an example, consider again the 2-scoped run from Figure 1. Note that T_1 and T_2 are canonical thread-interfaces for it, and $T_1 = (\langle q_0, q_3\rangle, \langle q_5, q_8\rangle) \bowtie_2 (\langle q_8, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$ and $T_2 = (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle) \bowtie_1 (\langle q_{15}, q_{17}\rangle)$ (the interfaces used in the compositions are all of dimension at most 2).

The above result is formally stated in the following lemma.

▶ **Lemma 4.** *Let $k \in \mathbb{N}$, M be an n-MPDS, ρ be a k-scoped run of M, and I be a canonical i-thread-interface for ρ, $i \in [n]$.*

There exist i-thread-interfaces $I_0, \ldots, I_s$ of dimension at most k such that $I = I_0 \bowtie_{j_1} I_1 \ldots \bowtie_{j_s} I_s$ with $j_1, \ldots j_s \in [2]$.

For i=1,2, let $I_i = \langle in_j^i, out_j^i \rangle_{j \in [r_i]}$ be a thread-interface of M. We say that I_1 *stitches to* I_2 *up to index* r (shortly, r-stitches) if $r \leq \min\{r_1, r_2\}$, and $out_j^1 = in_j^2$ for every $j \in [r]$. Also, I_2 *wraps with* I_1 *up to index* r (shortly, r-wraps) if $r \leq \min\{r_2, r_1 - 1\}$ and $out_j^2 = in_{j+1}^1$ for every $j \in [r]$. Note that, in particular, if either I_1 or I_2 is empty (i.e., dimension is 0), I_1 does not r-stitch to I_2 and I_1 does not r-wrap with I_2 for any $r > 0$.

In our running example, T_1 3-stitches to T_2 and T_2 2-wraps with T_1, and the two interfaces correspond to the run in Figure 1, and similarly the pair T_1, T_4 corresponds to the run portion from q_0 through q_{15}.

In general, we can show that runs of MPDS can be fully characterized by tuples of thread-interfaces. In fact, by definition, each m-round run of an n-MPDS M exactly defines a canonical thread-interface I_i where $i \in [n]$ such I_i m-stitches to I_{i+1}, for every $i \in [n-1]$, and I_n $(m-1)$-wraps with I_1. Vice-versa given the i-thread-interfaces I_i with $\dim(I_i) = m$, $i \in [n]$, such that I_i m-stitches to I_{i+1}, for every $i \in [n-1]$, and I_n $(r-1)$-wraps with I_1, from the definition of thread-interface we can construct an m-rounds run of M by concatenating the runs corresponding to each interface. Also, observe that $I_1, \ldots, I_n$ are the canonical thread-interfaces of the constructed run, and for $j \in [m]$, their j'th pairs contain the states at which the constructed run context-switches in round j. Thus, we get the following lemma.

▶ **Lemma 5.** *Let M be an n-MPDS, and q be an M control state. Then, there is a run of M reaching q iff there are n thread-interfaces $I_1, I_2, \ldots, I_n$ all of dimension r, where $I_i = \langle in_j^i, out_j^i \rangle_{j \in [r]}$ is a i-thread-interface of M, such that*

- *I_i r-stitches to I_{i+1}, for every $i \in [n-1]$;*
- *I_n $(r-1)$-wraps with I_1;*
- *in_1^1, the first input state of I_1, is the initial state of M and*
- *out_r^n, the last output state of I_n, is q.*

4 Fixed-Point Algorithm for Scope-Bounded Reachability

In this section, we describe a new algorithm to solve the scope-bounded reachability problem for MPDS. We recall that this problem has been recently shown to be decidable in [15]. Besides the differences in the approach, the solution we present here is fixed-point and has several advantages. First, our algorithm has a direct implementation in the tool GETAFIX [11]. Moreover, from our fixed-point characterization, we can easily derive a straightforward sequentialization algorithm, as well as prove that multiply nested words of scope-bounded runs have bounded tree-width, and therefore, a number of properties (including scope-bounded reachability) can be shown to be decidable by Courcelle's theorem (see Section 5). We start defining the scope-bounded reachability problem.

▶ **Definition 6** (SCOPE-BOUNDED REACHABILITY PROBLEM)**.** Fix $k \in \mathbb{N}$. For an n-MPDS M and an M control state q, the *k-scoped reachability problem* asks whether there is a k-scoped run of M from an initial configuration to any configuration of the form $\langle q, \{w_i\}_{i \in [n]} \rangle$. ◀

The algorithm. One way to solve the scope-bounded reachability problem is to first non-deterministically compute n thread-interfaces of the same dimension, one for each thread, and then by Lemma 5 check whether they form an M computation reaching state q. The drawback of this approach is that it gives a semi-algorithm as we do not know, a priori, the number of rounds that would be needed to conclude that q is not reachable. In contrast, the solution we propose, as a fixed-point algorithm, would implement the same approach as outlined above with the difference that we do not generate thread interfaces one after another, but rather in parallel, as the components of a tuple.

At each step, we append via the operators $\bowtie_1$ and $\bowtie_2$ a thread-interface to the component of the tuple that has the least dimension (*thread-interface progression rule*). In doing so, we also check that the appended thread-interface is compatible with the rest of the tuple (i.e., it appropriately stitches to its neighbours in the tuple and for the first/last component it wraps with the opposite end-tuple).

By computing the canonical thread-interfaces this way, as soon as the pairs of states of index j have been added to all components of the tuple, all of them can be safely removed (provided that if the first component does not have the pair of index $j+1$, we store the *out*-state removed from the last component). This corresponds to advancing the starting round of the tuple to the next round in a run matching the tuple of canonical thread-interfaces (*round deletion rule*).

To minimise the size of the components in the stored tuples, we apply the round deletion rule with higher priority than the thread-interface progression rule. This way, we can keep the dimension of each tuple component not larger than k, and this ensures also the convergence of the algorithm.

Our algorithm maintains tuples of the form $\nu = [I_1, \ldots, I_n]$ where each I_i is a *fragment* of an i-thread-interface: if ν is computed by our algorithm, then there exist n canonical thread-interfaces $I'_1, \ldots, I'_n$, where I'_i is an i-thread interface, which satisfy the conditions of Lemma 5, and furthermore, $I'_i \bowtie_1 I_i$ is an i-thread-interface. Note that each I_i may not be a thread-interface.

We implement the above mentioned priority by firing the thread-interface progression rule only on tuples containing a *basic* fragment of thread interface. For a tuple $\nu = [I_1, \ldots, I_n]$, I_j is *basic* if it is empty, or has dimension 1 and does not match completely the corresponding execution context in ρ (in particular, it matches the state when context-switching into such context but does not match the state when context-switching out of it). Formally, we say that I_j, $j \in [n]$, is *basic for* $[I_1, \ldots, I_n]$ if either one of the following conditions holds:

(*i*) $I_j = \emptyset$, *or*

(*ii*) $j < n$, $\dim(I_j) = 1$, $I_{j+1} \neq \emptyset$, and I_j does not 1-stitch to I_{j+1}, *or*

(*iii*) $j = n$, $\dim(I_n) = 1$, $\dim(I_1) > 1$, and I_n does not 1-wrap with I_1.

A thread-interface $I = \langle in_j, out_j \rangle_{j \in [r]}$ is *initial* if $r > 0$ and $in_1 = q_0$. Denoting with $\emptyset$ the empty thread-interface, the set of tuples computed by the algorithm, denoted $\mathcal{I}$, is initialized to all n-tuples $[I, \emptyset, \ldots, \emptyset]$ where I is an initial 1-thread-interface.

The detailed rules of the algorithm are given in Figure 2. There, we have denoted with $\bowtie$ the extension of $\bowtie_2$ such that $I \bowtie J$ is J, if $I = \emptyset$, and $I \bowtie_2 J$, otherwise. Note that $\bowtie$ is defined as $\bowtie_1$ when the first argument is $\emptyset$, and thus captures the composition of thread-interfaces via $\bowtie_1$ in the thread-interface progression rule. Also, in the thread-interface progression rule we do not force the matching on the last index value for J. This is to capture the cases when the composition of the canonical thread-interface requires the use of the $\bowtie_2$ operator. Finally, observe that for tuples where the 1-thread-interface has dimension 1, with the round deletion rule we do not simply delete this thread-interface but we replace it with the 1-thread-interface $(\langle out_1, out_1 \rangle)$ where out_1 is the out-state of the first pair of the last fragment in the tuple. The reason we handle the first thread differently from the others resides in the fact that the matching of state out_1 with the corresponding in-state of the 1-thread-interface (wrapping condition) cannot be checked at this time since this thread-interface has dimension 1. Therefore, it is necessary to store it for future matching.

The thread-interfaces of dimension at most k can be computed in a standard way, see for example [11], and thus we omit it. The algorithm halts when no more tuples can be added to the set $\mathcal{I}$.

Initialization.
$\mathcal{I} = \{[I, \emptyset, \ldots, \emptyset] \mid I \text{ is an initial 1-thread-interface of } M \text{ with } \dim(I) \leq k\}$

Thread-interface progression—append thread-interface to basic thread-interface fragment.
`Let` $[I_1, \ldots, I_n] \in \mathcal{I}$ and for some $j \in [n]$, I_j be basic for $[I_1, \ldots, I_n]$.
`Add` to $\mathcal{I}$ any tuple obtained from $[I_1, \ldots, I_n]$ by replacing I_j with $J = I_j \bowtie J'$ such that:
1. J' is a j-thread-interface of M of dimension $s \leq k$,
2. (For every $h \in [n]$, let r_h be the dimension of I_h)
   - `if` $j = 1$ `then` I_n r-wraps with J for $r = \min\{s, r_n - 1\}$
   - `if` $j > 1$ `then` I_{j-1} r-stitches to J for $r = \min\{s, r_{j-1} - 1\}$
   - `if` $j < n$ `then` J r-stitches to I_{j+1} for $r = \min\{s - 1, r_{j+1}\}$
   - `if` $j = n$ `then` J r-wraps with I_1 for $r = \min\{s - 1, r_1\}$

Round deletion—remove completed rounds from tuples.
`Let` $[I_1, \ldots, I_n] \in \mathcal{I}$ be s. t. I_j 1-stitches to I_{j+1} for $j \in [n-1]$
and either: (a). I_n 1-wraps with I_1 or (b). $\dim(I_1) = 1$.
`Add` $[I_1'', \ldots, I_n'']$ to $\mathcal{I}$ such that:
1. for $j \in [2, n]$, $I_j = I_j' \bowtie_1 I_j''$ and $\dim(I_j') = 1$
2. `if` (a) holds `then` also $I_1 = I_1' \bowtie_1 I_1''$ and $\dim(I_1') = 1$
   `else` (i.e., (b) holds) $I_1'' = (\langle out_1, out_1 \rangle)$, where $I_n = \langle in_j, out_j \rangle_{j \in [\dim(I_n)]}$.

Figure 2 Rules of the fixed-point algorithm solving the k-scoped reachability problem.

As an example, consider again the run of Figure 1. Our fixed-point algorithm computes the canonical thread-interface of the first thread as $(\langle q_0, q_3 \rangle, \langle q_5, q_8 \rangle) \bowtie_2 (\langle q_8, q_{10} \rangle, \langle q_{13}, q_{15} \rangle)$ and that of the second thread as $(\langle q_3, q_5 \rangle, \langle q_{10}, q_{13} \rangle) \bowtie_1 (\langle q_{15}, q_{17} \rangle)$. Thus, only fragments of dimension at most 2 are stored (3 is the dimension of the canonical thread-interfaces).

Transition system. The computation of the fixed-point algorithm described above on an n-MPDS M naturally defines a finite-state nondeterministic transition system. The states of the system are the tuples of fragments of thread-interfaces of dimension at most k, and the initial states and the transitions are given by the rules in Figure 2.

Formally, we define the transition system $\mathcal{A}_M^k = (\mathcal{I}_0, \mathcal{I}, \delta)$ where $\mathcal{I}_0 = \{[I, \emptyset, \ldots, \emptyset] \mid I$ is an initial 1-thread-interface of M and $\dim(I) \leq k\}$ is the set of initial states, $\mathcal{I}$ is the set of states, and $\delta \subseteq \mathcal{I} \times \{1, 2\} \times \mathcal{I}$ is the transition relation and contains all tuples (ν, i, ν') such that ν' is obtained from ν by applying the thread-interface progression rule, if $i = 1$, and the round deletion rule, otherwise (i.e., $i = 2$). A *run* of $\mathcal{A}_M^k$ is any sequence $\pi = \nu_0 \xrightarrow{m_1} \nu_1 \xrightarrow{m_2} \ldots \xrightarrow{m_t} \nu_t$ such that ν_0 is initial, $(\nu_{j-1}, m_j, \nu_j) \in \delta$ for every $j \in [t]$, and ν_t is of the form $[(\langle q, q \rangle), \emptyset, \ldots, \emptyset]$ for some M state q.

Correctness. Fix a n-MPDS $M = (Q, q_0, \Gamma, \{(\delta_i^{int}, \delta_i^{push}, \delta_i^{pop})\}_{i \in [n]})$ and $k \in \mathbb{N}$. Given a run π of $\mathcal{A}_M^k$, let $J_1, \ldots, J_m$ be the sequence of thread-interfaces that are used in the application of the thread-interface progression rule along π (transitions labeled with 1) in the ordering they appear in π. Furthermore, we assume that J_i is appended to the j_i component of the state. With $\mathit{Tuple}(\pi)$ we denote the tuple $[I_1, \ldots, I_n]$ that is obtained starting from ν_0 (the first state of π) by iteratively appending for $i = 1, \ldots, n$, J_i to the j_i'th component via $\bowtie_1$ if J_i replaces the $\emptyset$ by the effect of the corresponding transition, and $\bowtie_2$, otherwise.

By Lemma 4, we can show the following:

▶ **Lemma 7.** *$[I_1, \ldots, I_n]$ is the n-tuple of canonical thread-interfaces of a k-scoped computation of M iff there is a run π of $\mathcal{A}_M^k$ such that $\mathit{Tuple}(\pi) = [I_1, \ldots, I_n]$.*

Let $[I_1, \ldots, I_n]$ be the tuple of canonical thread-interfaces of a k-scoped run of M. By following the decomposition of each I_j given by Lemma 4 and then deleting rounds via transitions labeled with 2, it is possible to show that there is a run π of $\mathcal{A}^k_M$ such that $Tuple(\pi) = [I_1, \ldots, I_n]$, which ends in a state $[(\langle out, out\rangle), \emptyset, \ldots, \emptyset]$, where out is the last out-state of I_n.

Therefore, since the set $\mathcal{I}$ computed by the fixed-point algorithm given earlier in this section is also the set of states of $\mathcal{A}^k_M$, by Lemmas 5 and 7 we get:

▶ **Theorem 8.** *Let M be an n-MPDS, q be an M control state, and $k \in \mathbb{N}$. Then, q is reachable in a k-scoped computation of M iff $[(\langle q, q\rangle), \emptyset, \ldots, \emptyset] \in \mathcal{I}$.*

Sequentialization. It is possible to construct a pushdown system (1-MPDS) P^k_M such that the scope-bounded reachability problem on a given n-MPDS M can be reduced to standard reachability on P^k_M. The pushdown system P^k_M is essentially obtained by composing the transition system $\mathcal{A}^k_M$ with the threads M_i such that each transition $(\nu, 1, \nu')$ of $\mathcal{A}^k_M$ involving an i-thread-interface is replaced by a computation of M_i that computes this i-thread-interface followed by a thread-switch.

By Theorem 8, we can show the following:

▶ **Theorem 9.** *Let M be an n-MPDS and $k \in \mathbb{N}$. Then, the k-scoped reachability for M can be reduced to reachability for the pushdown system P^k_M.*

5 Tree-width of bounded scoped multiply nested words

In this section, we show that the set of multiply nested words corresponding to k-scoped runs of any n-MPDS has tree-width bounded by $2kn$.

To each k-scoped computation ρ of an n-MPDS M we associate a labelled graph nw^ρ, called the *multiply nested word* of ρ, as follows. Let $\rho = C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_m} C_m$. Then, $nw^\rho = (V, E_L, \{E_h\}_{h\in[n]})$ where $V = \{v_i \mid i \in [0, m]\}$ is the set of vertices of nw^ρ, $E_L = \{(v_{i-1}, v_i) \mid i \in [m]\}$ is the set of all linear edges, and E_h is the set of all edges (v_i, v_j) such that $\mu^\rho_h(i, j)$ holds true. Figure 3 shows the multi-nested word of the run of Figure 1.

▶ **Definition 10** (TREE-WIDTH)**.** A *tree-decomposition* of a graph $(V, E_1, \ldots, E_m)$ is (T, bag), where T is a binary tree with set of nodes N, and $bag : N \to 2^V$ s.t.

- For every $v \in V$, there is a node $n \in N$ such that $v \in bag(n)$,
- For every $(u, v) \in \bigcup_{i\in[m]} E_i$, there is a node $n \in N$ such that $u, v \in bag(n)$,
- If $u \in bag(n)$ and $u \in bag(n')$, for nodes $n, n' \in N$, then for every n'' that lies on the unique path connecting n and n', $u \in bag(n'')$.

The *width* of a tree decomposition (T, bag) is the size of the largest bag in it, minus one; i.e. $max_{n\in N}\{|bag(n)|\} - 1$. The *tree-width* of a graph is the *smallest* of the widths of any of its tree decompositions. ◀

The tree-width of bounded-scoped multiply nested words. We show that, for any k-scoped computation ρ of an n-MPDS M, the tree-width of the corresponding multiply nested words nw^ρ is bounded by $2nk$. For each nw^ρ, we describe a tree decomposition that uses as basic blocks the tree decompositions of the subgraphs corresponding to thread-interfaces and arrange

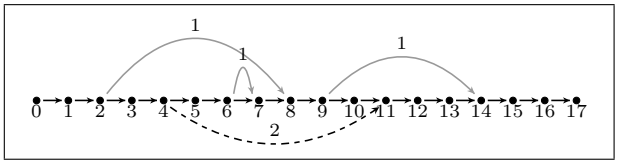


Figure 3 The 2-nested word of the run of Fig. 1.

them into a tree decomposition for the entire graph according to the corresponding run of the transition system $\mathcal{A}^k_M$ (defined in Section 4).

For the rest of the section, fix an n-MPDS M, a k-scoped run ρ of M and a run $\pi = \nu_1 \overset{m_1}{\longmapsto} \nu_2 \overset{m_2}{\longmapsto} \dots \overset{m_{t-1}}{\longmapsto} \nu_t$ of $\mathcal{A}^k_M$ such that $Tuple(\pi)$ is the tuple of canonical thread-interfaces of ρ. Also, denote $nw^\rho = (V, E_L, \{E_h\}_{h\in[n]})$.

Let J_1 be the 1-thread-interface of ν_1, and for $j \in [2, m]$, $\nu_{i_j-1} \overset{1}{\longmapsto} \nu_{i_j}$ be all the 1-transitions of π (i.e., those related to the application of the thread-interface progression rule) and J_j be the thread-interface there used. For $i \in [m]$, let $\rho^i_1, \dots, \rho^i_{r_i}$ be the portions of ρ that correspond to J_i. Note that, except for the starting and the ending configurations each ρ^i_j is disjoint from each other, and ρ can be constructed by stitching the ρ^i_j, one to another, on the starting and ending configurations in some order.

We define $G^i = (V^i, E^i_L, \{E^i_h\}_{h\in[n]})$ as the subgraph of nw^ρ over the vertices $V^i \subseteq V$ that correspond to the configurations visited in the runs $\rho^i_1, \dots, \rho^i_{r_i}$. Note that, E^i_L contains all the edges of E_L that connect two vertices of V^i, E^i_h is empty for $h \neq i$, $h \in [n]$, and E^i_i contains all the edges of E_i that connect two vertices of V^i. We denote with B^i the subset of V^i containing all the vertices that correspond to the starting and the ending configurations of each ρ^i_j, $j \in [r_i]$.

We observe that all the edges of nw^ρ except for those in G^i, $i \in [m]$, do not connect vertices in $V^i \setminus B^i$, and thus B^i contains all the vertices that connect G^i with the rest of the graph nw^ρ.

Given two graphs $G' = (V', E'_L, \{E'_h\}_{h\in[n]})$ and $G'' = (V'', E''_L, \{E''_h\}_{h\in[n]})$ the union of G' and G'', denoted $G' \cup G''$, is the graph $(V' \cup V'', E'_L \cup E''_L, \{E'_h \cup E''_h\}_{h\in[n]})$.

For all the above, clearly nw^ρ can be seen as the union of G^i for $i \in [m]$.

We recall that any subgraph G of a multi-nested word which corresponds to a thread-interface I of dimension k has a tree-decomposition of width at most $2k + 1$ [20]. In this decomposition, the bag of the root contains exactly the vertices corresponding to the starting and ending configurations of the runs corresponding to I, therefore its size is at most $2k$.

For each G^i, $i \in [m]$, denote with $TD_i = (T_i, bag_i)$ the tree-decomposition of G^i as in [20]. Observe that the bag of the root of each T_i is exactly B^i.

Now, define the sequence $\mathcal{B}_1, \dots, \mathcal{B}_t$ as follows. The element $\mathcal{B}_1$ is the set of vertices B^1. For each $i \in [2, t]$ such that $\nu_i = \nu_{i_j}$, i.e., in π the transition from ν_{i-1} to ν_i is an application of the thread-interface progression rule, we set $\mathcal{B}_i = \mathcal{B}_{i-1} \cup B^j$. Otherwise, i.e., the transition from ν_{i-1} to ν_i is an application of the round-deletion rule, we set $\mathcal{B}_i = \mathcal{B}_{i-1} \setminus D^i$, where with D^i we denote the vertices of $\mathcal{B}_{i-1}$ which correspond to the elements that are deleted from the fragments of thread-interface moving from ν_{i-1} to ν_i in π. Note that for $i \in [m]$, $|\mathcal{B}_i| \leq n(k + 2)$.

A tree-decomposition $TD = (T, bag)$ for nw^ρ is thus as follows (see Figure 4). The leftmost path of T corresponds to the sequence $\nu_1\nu_{i_2} \dots \nu_{i_m}$. Precisely, denoting with $u_1 \dots u_m$ the leftmost path of T, $bag(u_1) = \mathcal{B}_1$, and for $j \in [2, m]$, $bag(u_j) = \mathcal{B}_{i_j}$.

By the definition of $\mathcal{A}^k_M$, if a vertex v is in the bag of two nodes u_i and u_j, $i \leq j$, then v is also in all the bags of the nodes of the path $u_i u_{i+1} \dots u_j$.

The rest of TD is given by adding TD_j as right child of the nodes u_j, $j \in [m]$.

Recall that the edges outside of G^i cannot have as an end-point a vertex of G^i which is not in B^i.

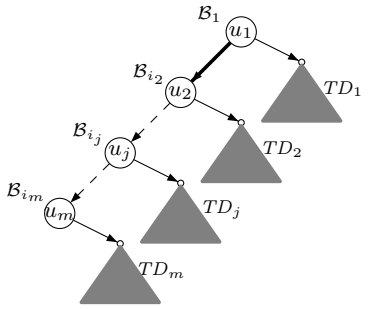


Figure 4 Tree-decomposition TD.

Moreover, we have that if a vertex v is in the bags
of two nodes on the leftmost path, then it is also in bags of all the nodes in the between. Therefore, since each subtree TD_i is a tree-decomposition and the bag of the root of each T_i is exactly B^i, we can conclude that TD is a tree decomposition for nw^ρ. Moreover, since TD_i has tree-width at most $2k$ and $|bag(u_i)| \leq n(k+2)$, for $i \in [m]$, we get that the tree-width of nw^ρ is at most $2kn$.

▶ **Lemma 11.** *For any $k, n \in \mathbb{N}$, the class of all k-scoped n-nested word graphs has tree-width bounded by $2nk$.*

Multiply nested word graphs are Monadic Second Order (MSO) definable (see [20]). Furthermore, the bounded scope restriction is easily expressible in MSO on multiply nested words. Thus, following the approach of [20] we have.

▶ **Theorem 12.** *The satisfiability problem of any MSO sentence on the class of all k-scoped n-nested word graphs is decidable.*

6 Conclusions and Future Work

We have presented a new algorithm for solving the reachability problem on scope-bounded Mpds. Our solution is fixed-point and allows a new sequentialization algorithm, which is useful for the analysis of concurrent programs by means of sequential verification tools. We have also shown that the class of multiply nested words for scope-bounded executions has bounded tree-width. Below we describe possible implications and further explorations that we believe it is worth to pursue.

Our fixed-point formulation for the reachability problem of scope-bounded Mpds has direct encoding in Getafix [11], an efficient verification tool for sequential and concurrent Boolean programs. It would be interesting to empirically evaluate our solution in Getafix on several abstractions of device drivers.

The sequentialization we propose can be extended to real programming languages and can be realised as a code-to-code translation from concurrent to sequential programs. We plan to implement this sequentialization for the C language by using the frama-C framework, and employ several sequential verification tools for the analysis, such as Corral [17] which has been optimized for sequentializations of concurrent programs.

Recently, Madhusudan and Parlato have shown that the reachability problem of several restrictions of Mpds is decidable by providing a uniform decidability schema [20]. In this paper we show that also the scope-bounded restriction fits in this framework. Furthermore, in [20], it is shown the decidability smoothly extends to any MSO property as well as to infinite runs. This allows us to get a series of new decidability results for scope-bounded Mpds, such as the decidability of Linear Temporal Logic and the concurrent temporal logic introduced in [16].

References

1 Mohamed Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. Linear-time model-checking for multithreaded programs under scope-bounding. In *ATVA*, volume 7561 of *LNCS*, pages 152–166, 2012.

2 Mohamed Faouzi Atig, Ahmed Bouajjani, and Gennaro Parlato. Getting rid of store-buffers in TSO analysis. In *CAV*, volume 6806 of *LNCS*, pages 99–115. Springer, 2011.

3 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. In *TACAS*, volume 5505 of *LNCS*, pages 107–123. Springer, 2009.

4 Ahmed Bouajjani and Michael Emmi. Bounded phase analysis of message-passing programs. In *TACAS*, volume 7214 of *LNCS*, pages 451–465. Springer, 2012.

5 Ahmed Bouajjani, Michael Emmi, and Gennaro Parlato. On sequentializing concurrent programs. In *SAS*, volume 6887 of *LNCS*, pages 129–145. Springer, 2011.

6 Ahmed Bouajjani, Séverine Fratani, and Shaz Qadeer. Context-bounded analysis of multithreaded programs with dynamic linked structures. In *CAV*, volume 4590 of *LNCS*, pages 207–220. Springer, 2007.

7 Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. Mso decidability of multi-pushdown systems via split-width. In *CONCUR*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.

8 Michael Emmi, Shaz Qadeer, and Zvonimir Rakamaric. Delay-bounded scheduling. In *ACM SIGPLAN-SIGACT POPL*, pages 411–422, 2011.

9 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE Computer Society, 2007.

10 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. An infinite automaton characterization of double exponential time. In *CSL*, volume 5213 of *LNCS*, pages 33–48. Springer, 2008.

11 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Analyzing recursive programs using a fixed-point calculus. In *PLDI*, pages 211–222. ACM, 2009.

12 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Reducing context-bounded concurrent reachability to sequential reachability. In *CAV*, volume 5643 of *LNCS*, pages 477–492. Springer, 2009.

13 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.

14 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Sequentializing parameterized programs. In *FIT*, volume 87 of *EPTCS*, pages 34–47, 2012.

15 Salvatore La Torre and Margherita Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.

16 Salvatore La Torre and Margherita Napoli. A temporal logic for multi-threaded programs. In *IFIP TCS*, volume 7604 of *LNCS*, pages 225–239. Springer, 2012.

17 Akash Lal, Shaz Qadeer, and Shuvendu K. Lahiri. A solver for reachability modulo theories. In *CAV*, volume 7358 of *LNCS*, pages 427–443. Springer, 2012.

18 Akash Lal and Thomas W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. In *CAV*, volume 5123 of *LNCS*, pages 37–51. Springer, 2008.

19 Akash Lal, Tayssir Touili, Nicholas Kidd, and Thomas W. Reps. Interprocedural analysis of concurrent programs under a context bound. In *TACAS*, volume 4963 of *LNCS*, pages 282–298. Springer, 2008.

20 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In *ACM SIGPLAN-SIGACT POPL*, pages 283–294, 2011.

21 Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

22 Dejvuth Suwimonteerabuth, Javier Esparza, and Stefan Schwoon. Symbolic context-bounded analysis of multithreaded Java programs. In *SPIN*, volume 5156 of *LNCS*, pages 270–287. Springer, 2008.

Scheduling with Setup Costs and Monotone Penalties

Rohit Khandekar[1], Kirsten Hildrum[2], Deepak Rajan[3], and Joel Wolf[2]

1 **Knight Capital Group, Jersey City, NJ, USA**
rkhandekar@gmail.com
2 **IBM T. J. Watson Research Center, Hawthorne, NY, USA**
{hildrum,jlwolf}@us.ibm.com
3 **Lawrence Livermore National Laboratory, Livermore, CA, USA**
rajan3@llnl.gov

Abstract

We consider single processor preemptive scheduling with job-dependent setup times. In this model, a job-dependent setup time is incurred when a job is started for the first time, and each time it is restarted after preemption. This model is a common generalization of preemptive scheduling, and actually of non-preemptive scheduling as well. The objective is to minimize the sum of any general non-negative, non-decreasing cost functions of the completion times of the jobs — this generalizes objectives of minimizing weighted flow time, flow-time squared, tardiness or the number of tardy jobs among many others. Our main result is a randomized polynomial time $O(1)$-speed $O(1)$-approximation algorithm for this problem. Without speedup, no polynomial time finite multiplicative approximation is possible unless $\mathbb{P} = \mathbb{NP}$.

We extend the approach of Bansal et al. (FOCS 2007) of rounding a linear programming relaxation which accounts for costs incurred due to the non-preemptive nature of the schedule. A key *new* idea used in the rounding is that a point in the intersection polytope of two matroids can be decomposed as a convex combination of incidence vectors of sets that are independent in both matroids. In fact, we use this for the intersection of a partition matroid and a laminar matroid, in which case the decomposition can be found efficiently using network flows. Our approach gives a randomized polynomial time offline $O(1)$-speed $O(1)$-approximation algorithm for the broadcast scheduling problem with general cost functions as well.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Scheduling, resource augmentation, approximation algorithm, preemption, setup times

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.185

1 Introduction

In this paper, we consider a very general preemptive scheduling problem with job-dependent setup times. This model captures the necessity of performing a setup whenever a job is started for the first time, or restarted after being preempted. Such a setup time might be needed for a variety of practical reasons, such as loading the job context or acquiring the necessary resources. Furthermore, we set as our goal the minimization of the sum of arbitrarily given non-decreasing cost functions of the completion times of the jobs. (For this paper we will restrict our attention to non-negative cost functions.) This problem is general enough to capture several interesting min-sum cost functions such as weighted flow-time,

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 185–198

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

flow-time squared, tardiness or the number of tardy jobs. One can also encode min-max cost functions such as makespan or maximum stretch by doing binary searches on the optimum cost and setting job deadlines appropriately.

We now define our problem, which can be classified as $1 \mid pmtn, r_j, setup = s_j \mid \sum f_j(C_j)$. We call this problem a *general scheduling problem* or GSP. Let $\mathbb{Z}_+$ denote the set of non-negative integers, and $\mathbb{R}$ the set of all reals. Consider a set J of jobs, where each job $j \in J$ is associated with release time $r_j \in \mathbb{Z}_+$, setup time $s_j \in \mathbb{Z}_+$, processing time $p_j \in \mathbb{Z}_+$ and a non-decreasing cost function[1] $f_j : \mathbb{Z}_+ \to \mathbb{Z}_+ \cup \{\infty\}$. A feasible schedule on a single processor works on no job before its release time and works on at most one job at any time. The jobs can be preempted. However, every time a job j is started or restarted, a setup time of s_j must be spent before processing can begin. Thus, without loss of generality, any schedule which starts or restarts job j works on it for at least s_j time before preempting or completing it. A job j requires a total of p_j time for processing.[2] Thus if job j is preempted k times before it completes, the total amount of time (including time for setup and processing) it requires is $(k+1) \cdot s_j + p_j$. Given a feasible schedule, let C_j denote the completion time of job j. The objective is to find a feasible schedule that minimizes the total cost $\sum_{j \in J} f_j(C_j)$.

The above problem generalizes both preemptive scheduling (when $s_j = 0$ for all j) and non-preemptive scheduling (when $p_j = 0$ for all j). As a result, obtaining small multiplicative approximation factors for GSP in polynomial time can be ruled out. More precisely, it is impossible to obtain an $n^{1/2-\epsilon}$-approximation for any $\epsilon > 0$ in polynomial time even for minimizing non-preemptive unweighted flow-time (i.e., $f_j(t) = \max\{0, t - r_j\}$) on n-job instances unless $\mathbb{P} = \mathbb{NP}$ [11].

A commonly used method for dealing with such problems is *resource augmentation analysis*, first proposed by Kalyanasundaram and Pruhs [10] and named so by Phillips et al. [12]. In this methodology, one compares the candidate algorithm, equipped with a faster processor, to an optimum algorithm with a unit speed processor. Define an s-speed ρ-approximation algorithm to be one which, using a processor of speed s, can achieve an objective value no more than ρ times the optimum value on a processor of speed 1. One also considers an analogous notion of extra processors instead of or in addition to extra speed: an m-processor s-speed ρ-approximation algorithm is one which, using m processors of speed s each, can achieve an objective value no more than ρ times the optimum value on a single processor of speed 1. This method of analysis can help elucidate the problem structure. For example, it can be used to explain why some algorithms work well in practice. It can also be used to explain why hardness proofs fall apart when hard instances are perturbed even slightly, for example a reduction from 3-Partition that shows non-preemptive flow-time is hard. We refer the reader to Bansal et al. [3] for further explanation. In this paper, we use resource augmentation analysis.

1.1 Our Results

We summarize our results now. The main result is given in Theorem 1.

▶ **Theorem 1.** *There exists a randomized polynomial time $O(1)$-speed $O(1)$-approximation algorithm for* GSP.

[1] We assume that f_j is given by a value oracle that given $t \in \mathbb{Z}_+$ returns $f_j(t)$.

[2] In the case with $p_j = 0$ for some job j, we insist that job j must get its setup time s_j contiguously at least once.

▶ **Lemma 2.** *For any $\epsilon > 0$, there exists a randomized polynomial time $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})(1+\epsilon)$-approximation for preemptive scheduling ($s_j = 0$ for all j) and a randomized polynomial time 12-speed $2(1+\epsilon)$-approximation for non-preemptive scheduling ($p_j = 0$ for all j).*

▶ **Theorem 3.** *There exists a randomized polynomial time $O(1)$-speed $O(1)$-approximation algorithm for the broadcast version of* GSP.

To determine if there exists a randomized polynomial time $(1+\epsilon)$-speed $f(\epsilon)$-approximation algorithm for GSP for any $\epsilon > 0$, where $f(\epsilon)$ is any computable function of ϵ alone, is an interesting open question. However it is easy to show that a speedup, greater than 1, is needed to obtain any finite multiplicative approximation, even for the special case of non-preemptive scheduling to minimize the number of tardy jobs, if $\mathbb{P} \neq \mathbb{NP}$, as shown by the lemma below.

▶ **Lemma 4.** *Consider a special case of non-preemptive scheduling (i.e., $p_j = 0$ for all $j \in J$) to minimize the number of tardy jobs (i.e., $f_j(t) = 0$ if $t \leq d_j$, and 1 otherwise for deadline $d_j \in \mathbb{Z}_+$). It is strongly $\mathbb{NP}$-hard to distinguish between the instances that have zero optimum cost and the instances that have positive optimum cost.*

The definition of broadcast scheduling problem and proofs of Lemma 2, Theorem 3 and Lemma 4 are omitted from this version due to lack of space.

1.2 Our Techniques

Our algorithm is based on rounding a linear programming relaxation. Our LP relaxation and overall approach are motivated by the work of Bansal et al. [3], who consider min-sum non-preemptive scheduling problems like weighted flow-time, weighted tardiness and *unweighted* number of tardy jobs on single processor. The LP has a time-indexed formulation that pays a job-dependent setup time in each fractional processing of a job, and also gets partial credit for completing jobs fractionally. The LP also has the obvious constraints to ensure that each job is scheduled to the full extent and that at most one job is scheduled at any single time. Apart from these, the LP has one more crucial set of constraints used to lower bound the cost of any feasible schedule. These constraints are similar to those used by Bansal et al. [3] and are based on the following non-preemptive nature of the problem. Consider a job k that the LP decides to schedule continuously in a certain time interval $[t, t+\ell)$. Then any job j that is released in the interval $[t, t+\ell)$ must start no earlier than $t+\ell$. Thus it must pay at least $f_j(t+\ell)$ cost in the objective function. These constraints provide a good lower bound that a rounding scheme can charge against when the penalty function f_j of job j increases significantly between r_j and $t+\ell$.

1.2.1 New: rounding using total unimodularity of network flows

The main technical contribution of this paper as compared to Bansal et al. [3], however, is in rounding the fractional LP solution. The rounding scheme of Bansal et al. [3] works only when the fractional solution is so-called *laminar*. Intuitively, being laminar means that if the fractional schedule "preempts" job j in favor of scheduling job k, it starts processing job j again only after finishing job k. Such a property holds, for example, when there exists a total ordering $\prec$ on the jobs, such that $j \prec k$ iff the partial credit that the fractional solution gets by scheduling job k is at least that for job j at any point in time. Such an ordering exists for the case of weighted flow-time: $j \prec k$ iff k has higher weight than j, or if they have same weight and k is released before j.

The fractional solution may not be laminar, however, for arbitrary non-decreasing cost functions. Consider, for example, a cost function that encodes weighted completion time and a strict deadline, so that $f_j(t) = \omega_j t$ if $t \leq d_j$ and ∞ otherwise. Suppose two jobs j and k have a cost function of this form with release dates $r_j < r_k$, weights $\omega_j < \omega_k$ and deadlines $d_j < d_k$. In the absence of any other jobs, the fractional solution schedules job j starting at r_j. Once job k is released, it preempts job j in favor of job k because of the partial credit due to $\omega_k > \omega_j$. At some point later, it preempts job k in favor of job j to finish it by its deadline. It then schedules job k again. Thus the fractional solution is not laminar. There may not be a general way to massage such a solution to make it laminar without increasing its cost by too much.

Our approach works even if the fractional solution is *not* laminar. It first partitions the time into "aligned" intervals of length a power of β, an integer greater than 1. Thus these intervals are of the form $[a \cdot \beta^c, (a+1) \cdot \beta^c)$ where a and c are integers. We refer to c as the *class* of an interval of this type. It is easy to see that aligned intervals from all classes form a laminar family.[3] Intuitively speaking, our algorithm uses a rounding procedure on bipartite graphs where jobs on one side are fractionally assigned to aligned intervals from a certain class on the other side. We would like to convert this assignment into an integral assignment randomly while preserving the expectation and ensuring that the maximum number of jobs in any aligned interval is at most the ceiling of its expectation. We reduce this problem to rounding a fractional max-flow to an integral max-flow in a network with integral arc capacities. This can be done using the total unimodularity of network flow matrices, see details in Section 2.4. Our rounding is reminiscent of the bipartite graph based dependent rounding of Gandhi et al. [8]. Their rounding, however, cannot be used here, because our problem cannot be formulated as a rounding problem on bipartite graphs since we need to satisfy the so-called "degree-preservation constraints" for all aligned intervals which form a laminar family.

From a more general perspective, one can see this rounding as decomposing a point in the intersection polytope of a partition matroid and a laminar matroid as a convex combination of incidence vectors of sets which are independent in both the matroids. It turns out that the intersection polytope of a partition and a laminar matroid can be expressed as the set of feasible source-sink flows in a network with integral arc capacities. Thus the problem of decomposing a point in such a polytope as a convex combination of integral extreme points can be reduced to network flow computations. An algorithm, for rounding a point in the intersection polytope of two matroids, with some concentration properties was recently presented by Chekuri et al. [6]. We do not need their complex rounding scheme since we do not need the concentration properties in our analysis.

1.3 Related Work

The closest works to ours are Bansal et al. [3] and Bansal and Pruhs [4]. As mentioned before, Bansal et al. [3] consider *non-preemptive* single processor scheduling for min-sum objectives like weighted flow-time, weighted tardiness, *unweighted* number of tardy jobs. We generalize their rounding procedure to work with arbitrary cost functions. Bansal and Pruhs [4] consider single processor scheduling with the same general cost functions as we do, plus preemption and release dates. There are, however, no setup times. Reducing this problem to a particular geometric set-cover problem yields a randomized polynomial time

[3] A laminar family is not to be confused with a non-laminar fractional solution.

algorithm with approximation ratio $O(\log\log(nP))$, where P is the maximum job size. They also give an $O(1)$ approximation in the special case of identical release times. Recently, Im et al. [9] showed that the Highest-Density-First algorithm is $(2+\epsilon)$-speed $O(1)$-competitive for general monotone penalty functions. On the one hand, their algorithm is *online*, which is stronger, but on the other hand, they allow job preemption without any setup time penalty.

Several models for preemption penalties have been considered before. These include sequence-dependent, job-dependent or processor-dependent penalties. See Potts and van Wassenhove [13] and Allahverdi et al. [2] for surveys of the area. Most of the results deal with specific cost functions such as total completion time, total flow-time, or makespan. Schuurman and Woeginger [14], for example, present $(4/3+\epsilon)$-approximation for minimizing makespan in the context of multiple parallel processors, with preemption, job-migration and job-dependent setup times. There has also been some work in online scheduling with preemption penalties as well. For example, Divakaran and Saks [7] consider the single processor online problem with release times and setup times. The goal is to minimize the *maximum* flow time for a job. They present an $O(1)$-competitive algorithm for this problem. They also show that the offline problem is $\mathbb{NP}$-hard. Chan et al. [5] study the online flow time scheduling in the presence of preemption overheads and present a simple algorithm that is $(1+\epsilon)$-speed $(1+1/\epsilon)$-competitive.

2 Algorithm for gsp

2.1 Outline of the Algorithm

We give a randomized polynomial time $O(1)$-speed $O(1)$-approximation algorithm. Our algorithm and analysis have the following high-level steps.

1. Incurring a constant speedup factor, we argue that GSP can be reduced to a problem called *multi-piece non-preemptive scheduling* problem (MPSP) and one can restrict the search to so-called "aligned" schedules.
2. We then create and solve an LP relaxation to lower bound the cost of the optimum schedule.
3. We next perform randomized rounding of the fractional solution using network flow techniques to obtain a "pseudo"-schedule.
4. Losing another constant speedup factor, we convert the pseudo-solution into a feasible schedule.
5. Finally, incurring yet another constant speedup factor, we get a feasible solution with cost at most constant times the LP lower bound.

We remark that steps 1, 2 and 4 are very similar to the corresponding steps of the algorithm of Bansal et al. [3]. Our main contribution is in step 3. Step 5 is a final (and simple) wrap-up needed to bound the cost of the computed solution.

2.2 Step 1: Reduction to mpsp and Restricting to Aligned Schedules

We begin by reducing our problem to *multi-piece non-preemptive scheduling* problem (MPSP), defined as follows. The input to MPSP consists of a set J of jobs, where each job $j \in J$ is associated with release time $r_j \in \mathbb{Z}_+$, number of pieces $n_j \in \mathbb{Z}_+$, processing time $p_j \in \mathbb{Z}_+$ of each piece and a non-decreasing cost function[4] $f_j : \mathbb{Z}_+ \to \mathbb{Z}_+ \cup \{\infty\}$. A feasible schedule on

[4] We again assume that f_j is given by a value oracle that given $t \in \mathbb{Z}_+$ returns $f_j(t)$.

a single processor works on no job before its release time and works on at most one job at any time. A feasible schedule also schedules each job j in exactly n_j intervals, each of length exactly p_j. Given a feasible schedule, let C_j denote the completion time of job j, i.e., the maximum end time of any interval corresponding to job j. The MPSP is to find a feasible schedule that minimizes the total cost $\sum_{j \in J} f_j(C_j)$.

▶ **Lemma 5.** *If there is a polynomial-time σ-speed ρ-approximation algorithm for* MPSP*, there is a polynomial-time 2σ-speed ρ-approximation algorithm for* GSP*.*

Proof. Given instance $(J, \{(r_j, s_j, p_j, f_j) \mid j \in J\})$ of GSP, define an instance $(J, \{(r'_j, n'_j, p'_j, f'_j) \mid j \in J\})$ of MPSP by letting $r'_j = r_j$, $n'_j = \max\{1, \lceil p_j / \max\{1, s_j\} \rceil\}$, $p'_j = \max\{1, 2s_j\}$ and $f'_j = f_j$ for each job $j \in J$. Let OPT(GSP) denote the optimum value of the GSP instance and let OPT(MPSP) denote the optimum value of the MPSP instance on a processor with twice the speed. We first argue that OPT(MPSP) $\leq$ OPT(GSP). Consider the optimum schedule S of the GSP instance. We construct a feasible solution S' for the MPSP instance on a processor with twice the speed as follows. Consider an interval $[t, t + s_j + t')$ in which S schedules a job j. Let S' schedule $\lceil t' / \max\{1, s_j\} \rceil$ pieces of length $p'_j = \max\{1, 2s_j\}$ each in this interval. Since there is at least one such interval and the sum of processings t' over all such intervals is p_j, the total number of pieces scheduled for job j is at least n'_j. Therefore S' gives a feasible solution for the MPSP instance on a processor with twice the speed. It is easy to see that the cost of S' is at most that of S.

Now it is enough to show that given a solution S' for the MPSP instance, one can construct a solution S for the GSP instance feasible on a processor with the same speed and with cost at most that of S'. Consider an interval $[t, t + p'_j)$ in which S' schedules a piece of job j. Let S schedule job j in this interval so that it spends s_j time in setup and $\max\{1, s_j\}$ time in processing. Thus job j gets $n'_j \cdot \max\{1, s_j\} \geq p_j$ processing overall. It is easy to see that the cost of S is at most that of S'. ◀

In light of this lemma, we focus on designing an $O(1)$-speed $O(1)$-approximation algorithm for MPSP. Fix an instance $(J, \{(r_j, n_j, p_j, f_j) \mid j \in J\})$ of MPSP. Let $\beta > 1$ be an integer to be determined later. Incurring a speedup factor of β, we assume that all processing times in the instance are integer powers of β by replacing p_j by $\beta^{\lfloor \log_\beta p_j \rfloor}$. We define a notion of aligned schedules.

▶ **Definition 6.** We say that a schedule for MPSP is *aligned* if each piece of each job j is scheduled in an interval of the form $[ap_j, (a + 1)p_j)$ where $a \geq 0$ is an integer.

▶ **Lemma 7.** *On a processor with speedup factor 2, there exists an aligned schedule with cost at most that of the original* MPSP *instance.*

Proof. Fix an optimum schedule S for MPSP. Suppose S processes a piece of job j in the interval $[t, t + p_j)$. Let $t' \in [t, t + p_j/2)$ be an integral multiple of $p_j/2$. On a processor with twice the speed, we can process this piece of job j in the interval $[t', t' + p_j/2)$. It is easy to see that the resulting schedule is aligned and has cost at most that of S. ◀

To summarize, by incurring an overall speedup factor of 4β, we reduce GSP to MPSP, assume that each p_j is an integer power of β and set our goal to finding the aligned schedule with minimum cost. Let OPT denote the cost of an optimum aligned schedule of the new MPSP instance.

2.3 Step 2: Solving the Linear Programming Relaxation

We now present a linear programming relaxation for lower bounding OPT. Since any schedule, without loss of generality, completes all jobs by time $T = \max_j r_j + \sum_j n_j p_j$, it is enough to work within this time horizon. We introduce a variable $x(j,t)$ for each job j and each multiple t of p_j such that $t \geq r_j$. In the "intended" solution, $x(j,t) = 1$ if a piece of job j is scheduled in the interval $[t, t+p_j)$. We also introduce a variable W_j intended to lower bound the cost of job j.

$$\text{minimize} \quad \sum_j W_j \tag{1}$$

$$\forall j \in J, \quad n_j = \sum_t x(j,t) \tag{2}$$

$$\forall \tau \in \mathbb{Z}_+, \quad 1 \geq \sum_j \sum_{t:\tau \in [t,t+p_j)} x(j,t) \tag{3}$$

$$\forall j \in J, \quad W_j \geq \frac{1}{n_j} \sum_t x(j,t) \cdot f_j(t+p_j) \tag{4}$$

$$\forall j \in J, \quad W_j \geq \sum_{k \neq j} \sum_{t:r_j \in (t,t+p_k)} x(k,t) \cdot f_j(t+p_k) \tag{5}$$

$$\forall j,t, \quad x(j,t) \geq 0 \tag{6}$$

▶ **Lemma 8.** *The optimum value of the LP (1)-(6) is at most* OPT.

Proof. Consider an optimal aligned schedule S with cost OPT. Using S, we construct a feasible solution $\{x^*, W^*\}$ to the LP with value at most OPT. Let $x^*(j,t) = 1$ if S schedules a piece of j in the interval $[t, t+p_j)$ and 0 otherwise. Let $W_j^* = f_j(C_j)$ denote the cost of job j in S, where C_j denotes the completion time of j in S. It is easy to see that constraints (2) and (3) are satisfied by this solution, since each job j has exactly n_j pieces scheduled and S works on at most one job at a time, respectively. We now argue that the cost of job j in S is at least the right-hand-side of constraint (4) (the "average lower bound") or (5) (the "displacement lower bound") for job j. Fix a job j and let $[t_i, t_i + p_j)$ for $i = 1, 2, \ldots$ be the intervals containing the pieces of job j. The right-hand-side of constraint (4) for job j is the average of $f_j(t_i + p_j)$ for $i = 1, 2, \ldots$ which is clearly at most $W_j^* = \max_i f_j(t_i + p_j)$, the cost of job j in S. Thus the constraint (4) is satisfied. Now note that there is at most one job $k \neq j$ that has $x^*(k,t) > 0$ for values of t with $r_j \in (t, t+p_k)$. This is because S works on at most one job at a time. If such a job does not exist, it is easy to see that the constraint (5) is satisfied. If such a job k exists, the earliest time the first piece of job j can start is at least $t + p_k$. Thus its cost is at least $f_j(t+p_k)$. Since $x^*(k,t) = 1$ holds in such a case, the constraint (5) is satisfied. ◀

The number of variables and constraints in this LP is pseudo-polynomial. Using an approach similar to Bansal et al. [3], we can make this LP polynomial-sized at a small loss. We omit detailed from this version due to lack of space. For now, let us assume that we can compute a fractional optimum, denoted by $x^*(j,t)$ and W_j^* for $j \in J$ and $0 \leq t \leq T$, of this LP.

2.4 Step 3: Obtaining a Pseudo-Schedule via Network Flows

We say that a job j belongs to *class* $c \geq 0$ if $p_j = \beta^c$. Let J_c denote the set of jobs in class c. We obtain a pseudo-schedule for jobs in J_c for each class c separately. Fix a class c. To

obtain a pseudo-schedule for class c, we create an instance of the network flow problem. For an integer $d \geq 0$, we call an interval of the form $[a \cdot \beta^d, (a+1) \cdot \beta^d)$ an *aligned β^d-interval* or simply an *aligned interval*, where $a \geq 0$ is an integer.

2.4.1 Creating a flow network

Refer to Figure 1 for an example for the case of $\beta = 2$. The flow network we create is a layered directed acyclic graph with all arcs going between consecutive layers from top to bottom. The first layer consists of the `source` node. The second layer has a node v_j for each job $j \in J_c$. For $k \geq 0$, the $(k+3)$rd layer has a node v_I for each aligned β^{c+k}-interval I. The last layer has a node for the single aligned interval spanning the entire instance – we call this node the `sink`.

We add an arc from the `source` to v_j with capacity $n_j = \sum_t x(j,t)$, i.e., the total x-value job j receives. We add an arc from v_j to v_I for β^c-aligned interval $I = [t, t+\beta^c)$ such that $x(j,t) > 0$. We assign such an arc a capacity of

$$\lceil x(j,t) \rceil ,$$

i.e., the x-value that job j receives in aligned β^c-interval I rounded up to the nearest integer. For $k \geq 0$, we add an arc from v_I to $v_{I'}$ for an aligned β^{c+k}-interval I and aligned β^{c+k+1}-interval I' such that $I \subset I'$, provided v_I is not the `sink`. We give this arc a capacity of

$$\left\lceil \sum_{j \in J_c} \sum_{t:[t,t+\beta^c) \subseteq I} x(j,t) \right\rceil ,$$

i.e., the total x-value all jobs in J_c receive in aligned β^c-intervals contained in I rounded up to the nearest integer. Thus the flow network below layer 3 looks like an inverted β-ary tree. Note also that all arc-capacities in this network are integral.

Observe that the fractional solution $\{x(j,t) \mid j \in J_c, 0 \leq t \leq T\}$ gives a fractional feasible maximum flow in this network from `source` to `sink` of total value $\sum_{j \in J_c} n_j = \sum_{j \in J_c} \sum_t x(j,t)$ as follows. Send a flow of $n_j = \sum_t x(j,t)$ on $(\texttt{source}, v_j)$ for all $j \in J_c$, a flow of $x(j,t)$ on $(v_j, v_{[t,t+\beta^c)})$ for all $j \in J_c$ and t with $x(j,t) > 0$ and a flow of $\sum_{j \in J_c} \sum_{t:[t,t+\beta^c) \subseteq I} x(j,t)$ on the unique out-going arc $(v_I, v_{I'})$ for all aligned intervals I except the one corresponding to the `sink`.

We now use the fact that a network flow matrix is totally unimodular and hence the polytope of flows in a network with integral arc capacities has integral extreme points. Therefore any point inside such a polytope can be decomposed as a convex combination of integral flows.

▶ **Lemma 9.** *Consider a flow network with a source node, a sink node and integral arc capacities. Given a fractional maximum flow f in the network, one can compute a collection of integral maximum flows $F_1, \ldots, F_q$ and corresponding $\lambda_1, \ldots, \lambda_q \geq 0$ with $\sum_i \lambda_i = 1$ such that $f = \sum_i \lambda_i F_i$. Furthermore, the size q of the convex combination and its computation time is bounded polynomially in the input size.*

Proof. We cast the problem of decomposing the given maximum flow f as a convex combination of integral maximum flows as a linear program. Introduce a variable λ_i for each integral maximum flow F_i (that satisfies the arc capacity constraints). There may be exponentially

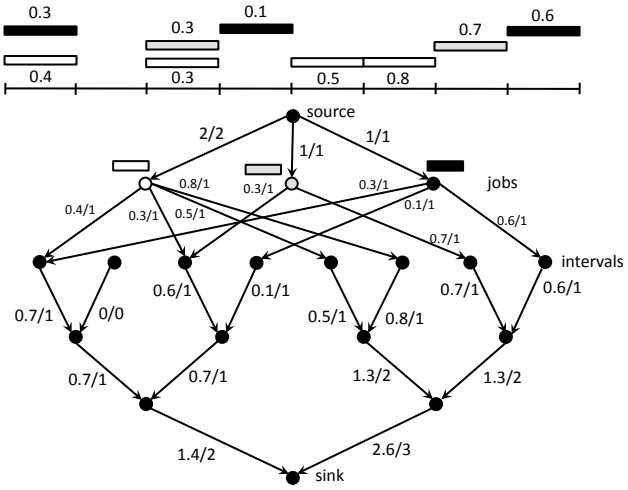


Figure 1 Creating a network flow instance. Consider the adjacent example with 3 jobs and 8 aligned intervals in a certain class. Assume that $\beta = 2$. The total x-values white, grey and black jobs receive are 2, 1 and 1 respectively. The corresponding flow network has a source node in the top layer, 3 nodes corresponding to jobs in the second layer and nodes corresponding to aligned intervals arranged in form of a β-ary tree. The bottom layer has a sink node. The numbers on the arcs denote their flows/integral capacities. The fractional maximum flow given is computed from the fractional solution.

many such flows. Now consider the following LP and its dual.

$$\text{Primal:} \quad \max\{\sum_i \lambda_i \mid \sum_i \lambda_i F_i(e) = f(e) \ \forall \text{ arcs } e, \lambda_i \geq 0 \ \forall \ i\},$$

$$\text{Dual:} \quad \min\{\sum_e f(e) l_e \mid \sum_e F_i(e) l_e \geq 1 \ \forall \ i, l_e \in \mathbb{R} \ \forall \text{ arcs } e\}.$$

The dual has exponentially many constraints but only polynomially many variables. We can solve the dual using the *ellipsoid* algorithm using the separation oracle that given not-necessarily-positive "arc-lengths" $\{l_e\}$ computes a maximum flow F_i with "minimum cost" $\sum_e F_i(e) l_e$. Several polynomial time algorithms exist for this well-known min-cost max-flow problem [1]. Recall that since all arc capacities are integral, this oracle returns an integral flow. The ellipsoid algorithm finds polynomially many maximum flows while computing the dual optimum solution. We can then restrict our attention to these maximum flows (i.e., set $\lambda_i = 0$ for all maximum flows F_i *not* found in the ellipsoid algorithm) and solve the new primal that now has polynomially many variables and constraints. Since each F_i as well as f are maximum flows, it is easy to see that the optimum primal solution thus computed has value $\sum_i \lambda_i = 1$. ◀

Our rounding procedure to obtain a pseudo-schedule for class c works as follows.

Procedure round: Use Lemma 9 to compute a convex combination of integral flows. Pick exactly one integral flow F_i with probability λ_i. Schedule $F_i(v_j, v_I)$ pieces of job j in the aligned β^c-interval I for all jobs j and aligned β^c-intervals I.

▶ **Lemma 10.** *The pseudo-schedule for all classes constructed by the above rounding procedure satisfies the following properties.*

1. *The expected number of pieces any job j receives in an aligned interval $[t, t+\beta^c)$ is $x(j,t)$.*
2. *With probability 1, each job j has exactly n_j pieces scheduled overall.*
3. *With probability 1, at most $\lceil \sum_{j \in J_c} \sum_{t:[t,t+\beta^c) \subseteq I} x(j,t) \rceil$ pieces of jobs in class c, counting multiplicities from the same job, are scheduled in any interval I of a class $d \geq c$.*
4. *Consider any interval in class d. With probability 1, the total size of all pieces of jobs in classes $0, \ldots, d$ scheduled in this interval is at most $\beta^d(2 + \frac{1}{\beta-1})$.*

Proof. The properties 1, 2 and 3 hold directly from the rounding. To see property 4, fix an interval I in class d. Summing the volume constraint (3) that the fractional solution satisfies over all $\tau \in I$

$$\sum_{c=0}^{d} \beta^c \left(\sum_{j \in J_c} \sum_{t:[t,t+\beta^c) \subseteq I} x(j,t) \right) \leq \beta^d.$$

Now from property 3, at most $\lceil \sum_{j \in J_c} \sum_{t:[t,t+\beta^c) \subseteq I} x(j,t) \rceil$ pieces of jobs in class c are scheduled in I. Therefore the total size of all the pieces of jobs in classes $0, \ldots, d$ scheduled in I is at most

$$\begin{aligned} \sum_{c=0}^{d} \beta^c \left\lceil \sum_{j \in J_c} \sum_{t:[t,t+\beta^c) \subseteq I} x(j,t) \right\rceil &< \sum_{c=0}^{d} \beta^c \left(1 + \sum_{j \in J_c} \sum_{t:[t,t+\beta^c) \subseteq I} x(j,t) \right) \\ &\leq \sum_{c=0}^{d} \beta^c + \beta^d \\ &= \beta^d \left(\frac{\beta}{\beta - 1} + 1 \right). \end{aligned}$$

◀

2.5 Step 4: Converting the Pseudo-Schedule into a Feasible Schedule

We use a factor $(2 + \frac{1}{\beta-1})$ speedup to convert the pseudo-schedule into a feasible schedule that schedules at most one job at any single time. Consider the pseudo-schedule produced in step 3. Call an aligned interval *maximal* if it contains a piece of a job of equal size and does not overlap with any other piece of a job of larger size. There can be multiple pieces corresponding to any maximal aligned interval. Fix a maximal interval I. We associate a natural β-ary tree corresponding to all the pieces overlapping with I. The tree-nodes in level d correspond to the aligned β^d-intervals overlapping with I. See Figure 2 for an example of such a tree for the case $\beta = 2$.

We give a procedure that uses a speedup factor of $(2+\frac{1}{\beta-1})$, and given a tree corresponding to a maximal interval I, feasibly schedules all the pieces in that tree in the aligned β^c-interval corresponding to the root. The schedule is feasible in the sense that pieces of each job are not scheduled before its release time and no two pieces overlap with each other. Since all the maximal intervals are non-overlapping, applying the above procedure to each corresponding tree produces a schedule for the entire instance.

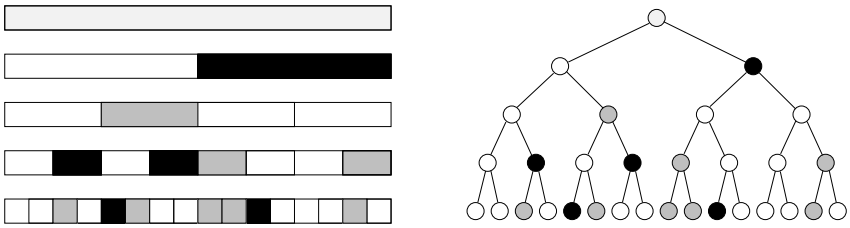


Figure 2 Example of a pseudo-schedule output by the rounding procedure for $\beta = 2$ and its corresponding β-ary tree. Each shaded box (and the respective tree-node) corresponds to one or more pieces of one or more jobs. From Lemma 10-4, the total size of pieces in any sub-tree (containing a node and all its descendants) is at most $(2 + \frac{1}{\beta-1})$ times the size of the root of the sub-tree. The black boxes/tree-nodes represent pieces of early jobs while grey boxes/tree-nodes represent pieces of late jobs. A box/tree-node can be both black or grey. (Source: Bansal et al. [3])

Procedure fit: We first shrink all pieces in J_I by a factor of $(2 + \frac{1}{\beta-1})$. We then compute the POSTORDER[5] traversal of the β-ary tree T_I. We schedule all pieces of early jobs in the order they appear in POSTORDER(T_I), pieces of equal lengths overlapping with each other ordered arbitrarily. We then compute the PREORDER traversal of T_I. We schedule all pieces of late jobs in the order they appear in PREORDER(T_I), pieces of equal lengths overlapping with each other ordered arbitrarily. These pieces of late jobs are then "right-justified", shifted as far right as possible so that the last piece completes at the end-point of interval I.

Consider a maximal interval $I = [\tau, \tau + \beta^c)$. Let J_I denote the set of jobs corresponding to pieces scheduled in the interval I, and let T_I denote the β-ary tree associated with the pseudo-schedule in I. We partition the jobs J_I into two sets, denoted *early* and *late*. The *early* jobs are the jobs $\{j \in J_I \mid r_j \leq \tau\}$ that are released not later than time τ. The pieces of these jobs can be scheduled anywhere in I. Note that even though an early job k is scheduled during the interval I, it does not pay the "penalty term" in the constraint (5). The *late* jobs are the jobs $\{j \in J_I \mid \tau < r_j < \tau + \beta^c\}$ that are released in I. A piece of a late job j can be scheduled no earlier than its release time r_j. Note that a late job j pays the "penalty term" in the constraint (5). We now describe our procedure FIT, to convert the pseudo-schedule into a feasible schedule. The following lemma shows that the schedule computed by the FIT procedure is feasible.

▶ **Lemma 11.** *The schedule output by the* FIT *procedure satisfies the following properties.*

1. *The pieces of jobs in J_I are scheduled in the interval I such that no two pieces overlap.*
2. *Each piece of each early job in J_I completes no later than its completion time in the pseudo-schedule.*
3. *Each piece of each late job in J_I starts no earlier than its start time in the pseudo-schedule, and it completes within I.*

[5] The POSTORDER traversal of a single-node tree v is defined as POSTORDER$(v) := v$ and that of a β-ary tree T with root r and left-to-right sub-trees $T_1, \ldots, T_\beta$ is recursively defined as POSTORDER$(T) :=$ POSTORDER$(T_1), \ldots,$ POSTORDER$(T_\beta), r$. Similarly, the PREORDER traversal of a single-node tree v is defined as PREORDER$(v) := v$ and that of a β-ary tree T with root r and left-to-right sub-trees $T_1, \ldots, T_\beta$ is recursively defined as PREORDER$(T) := r,$ PREORDER$(T_1), \ldots,$ PREORDER(T_β).

Proof. The first property follows from the observation that the total size of all the jobs in J_I is at most $(2 + \frac{1}{\beta-1})$ times the length of the interval I and the fact that we shrink all the pieces by a factor of $(2 + \frac{1}{\beta-1})$. We now prove the second property. Consider a piece π of an early job in J_I. Let its completion time in the pseudo-schedule be $\tau + \tau_\pi$. It is sufficient to argue that the total size of pieces of early jobs (including π) that come no later than π in POSTORDER(T_I) is at most $\tau_\pi(2 + \frac{1}{\beta-1})$ before shrinking. To this end, consider the prefix of POSTORDER(T_I) up to the tree-node corresponding to π. Let $T_1, \ldots, T_q$ be the disjoint subtrees of T_I that are traversed in POSTORDER(T_I) up to node π. Note that the root of T_q is π. Now let $I_1, \ldots, I_q$ be the (disjoint) intervals occupied by the roots of $T_1, \ldots, T_q$. Note that the total size of $I_1, \ldots, I_q$ is precisely τ_π. Furthermore, from Lemma 10-4, the total size of pieces of jobs in J_I that are contained in intervals $I_1, \ldots, I_q$ is at most $(2 + \frac{1}{\beta-1})$ times the total size of these intervals. Thus, in particular, the total size of pieces of the early jobs in these intervals is at most $\tau_\pi(2 + \frac{1}{\beta-1})$, and the property follows. The third property can be proved analogously, but with late jobs and PREORDER(T_I). ◀

2.6 Step 5: Final Wrap-up

The solution obtained in step 4 is feasible on a processor with speedup factor $4\beta(2 + \frac{1}{\beta-1})$. We now define pieces-wise average costs of a job $j \in J$ in the pseudo-schedule and a schedule computed by procedure FIT.

▶ **Definition 12.** Consider a job $j \in J$. Let $A_j = \frac{1}{n_j} \sum_{i=1}^{n_j} f_j(t_i + p_j)$ be the *average* cost of job j in the pseudo-schedule where $\{[t_i, t_i + p_j) \mid 1 \leq i \leq n_j\}$ denotes the set of intervals in which the pseudo-schedule computed by Procedure ROUND schedules job j. Furthermore let $A'_j = \frac{1}{n_j} \sum_{i=1}^{n_j} f_j(t'_i + p'_j)$ be the *average* cost of job j in the feasible schedule computed by Procedure FIT. Here $p'_j = p_j/(2 + \frac{1}{\beta-1})$ denotes the processing time of job j after the scaling and $\{[t'_i, t'_i + p'_j) \mid 1 \leq i \leq n_j\}$ denotes the set of intervals in which Procedure FIT schedules job j.

The following lemma bounds the expected average cost of a job in the feasible schedule.

▶ **Lemma 13.** *For any job $j \in J$, the expected value of A'_j is at most $2W_j$.*

Proof. From Lemma 10-1 and the constraint (4), it is clear that the expected value of A_j is at most W_j. We next prove that the expected value of $A'_j - A_j$ is at most W_j.

To this end, fix a job $j \in J$ and consider a piece $\pi_i = [t_i, t_i + p_j)$, where $1 \leq i \leq n_j$ in the pseudo-schedule. During Procedure FIT, job j was labelled as early for π_i with a certain probability and late for π_i with a certain probability. From Lemma 11, if job j was labelled as early for π_i, the piece π_i completed in the feasible schedule at or before its completion time in the pseudo-schedule and thus its contribution to $A'_j - A_j$ is non-positive. Now suppose that job j was labelled as late for π_i. Let $I_{\max}$ denote the random variable denoting the maximal interval that contains π_i. For any interval $I = [t, t + \ell) \ni r_j$, we have $I_{\max} = I$ with probability at most $\sum_{t,\ell: r_j \in (t,t+\ell)} \sum_{k: k \neq j, p_k = \ell} x(k,t)$. This follows from Lemma 10-1. In the event that $I_{\max} = I$, from Lemma 11, the piece π_i completes by time $t + \ell$. Thus the expected contribution of π_i to $A'_j - A_j$ is at most

$$\frac{1}{n_j} \sum_{t,\ell: r_j \in (t,t+\ell)} \sum_{k: k \neq j, p_k = \ell} x(k,t) \cdot f_j(t + \ell) = \frac{1}{n_j} \sum_{k \neq j} \sum_{t: r_j \in (t, t+p_k)} x(k,t) \cdot f_j(t + p_k).$$

Summing this over all pieces π_i of job j, we get from constraint (5) that the expected value of $A'_j - A_j$ is at most W_j. Hence the lemma holds. ◀

Note that the actual cost of job j is the maximum value of $f_j(t'_i + p'_j)$ over all of its pieces $\{[t'_i, t'_i + p'_j) \mid 1 \le i \le n_j\}$. For any non-decreasing cost function f_j, this cost can be arbitrarily larger than its average cost A'_j. Using a simple trick to bound the actual cost of the solution, we incur another factor α in speedup, where $\alpha > 1$ is an integer to be fixed later. We use the following simple observation. For a random variable x with range $\in \mathbb{Z}_+$ and a non-decreasing function $w : \mathbb{Z}_+ \to \mathbb{Z}_+$, we have $\mathbb{E}[w(x)] \ge \Pr[x \ge x_0] \cdot w(x_0)$ for all $x_0 \in \mathbb{Z}_+$, where $\mathbb{E}$ denotes the expectation operator.

Suppose the pieces $[t'_i, t'_i + p'_j)$ for $1 \le i \le n_j$ are sorted in the increasing order of their completion times $t'_i + p'_j$ in the feasible schedule. For any job j, shrink its processing time by factor α and schedule α pieces in each of the intervals $[t'_i, t'_i + p'_j)$ for $1 \le i < i_0 := \lceil n_j/\alpha \rceil$ and at most α pieces in the interval $[t'_{i_0}, t'_{i_0} + p'_j)$. It is easy to see that the actual cost of job j is at most

$$f_j(t'_{i_0} + p'_j) \le \frac{A'_j}{1 - \frac{\lfloor n_j/\alpha \rfloor}{n_j}} \le \frac{A'_j}{1 - \frac{1}{\alpha}} = \frac{\alpha A'_j}{\alpha - 1}.$$

Thus the expected actual cost of job j is at most $2\alpha W_j/(\alpha - 1)$.

In summary, we obtain a randomized algorithm that gives a feasible schedule on a processor with $4\beta(2 + \frac{1}{\beta - 1})\alpha$ speedup having expected cost at most $2\alpha/(\alpha - 1)$ times the optimum. If $\alpha = \beta = 2$, we get a randomized polynomial time 48-speed 4-approximation algorithm for MPSP, and (from Lemma 5) a randomized polynomial time 96-speed 4-approximation algorithm for GSP.

Acknowledgements. We thank Nikhil Bansal for useful discussions.

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
2. A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal on Operations Research*, 2008.
3. N. Bansal, H.-L. Chan, R. Khandekar, K. Pruhs, C. Stein, and B. Schieber. Non-preemptive min-sum scheduling with resource augmentation. In *Proceedings of the 48th Annual Symposium on Foundations of Computer Science*, pages 614–624, 2007.
4. N. Bansal and K. Pruhs. Geometry of scheduling. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science*, pages 81–90, 2004.
5. Ho-Leung Chan, Tak-Wah Lam, and Rongbin Li. Online flow time scheduling in the presence of preemption overhead. In *Proceedings of International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 85–97, 2012.
6. C. Chekuri, J. Vondrák, and R. Zenklusen. Multi-budgeted matchings and matroid intersection via dependent rounding. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1080–1097, 2011.
7. Srikrishnan Divakaran and Michael Saks. An online algorithm for a problem in scheduling with set-ups and release times. *Algorithmica*, 56, 2009.
8. Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.
9. Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Online scheduling with general cost functions. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1254–1265, 2012.
10. Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

11 H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 418–426, May 1996.

12 Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32:163–200, 2001.

13 C.N. Potts and L.N. van Wassenhove. Integrating scheduling with batching and lotsizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, 43:395––406, 1992.

14 Petra Schuurman and Gerhard Woeginger. Preemptive scheduling with job dependent-setup times. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 759–767, 1999.

Scheduling Resources for Executing a Partial Set of Jobs

Venkatesan T. Chakaravarthy[1], Arindam Pal[2], Sambuddha Roy[1], and Yogish Sabharwal[1]

1 **IBM Research Lab, New Delhi, India**
{vechakra,sambuddha,ysabharwal}@in.ibm.com
2 **Indian Institute of Technology, New Delhi.**
arindamp@cse.iitd.ernet.in

Abstract

In this paper, we consider the problem of choosing a minimum cost set of resources for executing a specified set of jobs. Each input job is an interval, determined by its start-time and end-time. Each resource is also an interval determined by its start-time and end-time; moreover, every resource has a capacity and a cost associated with it. We consider two versions of this problem.

In the partial covering version, we are also given as input a number k, specifying the number of jobs that must be performed. The goal is to choose k jobs and find a minimum cost set of resources to perform the chosen k jobs (at any point of time the capacity of the chosen set of resources should be sufficient to execute the jobs active at that time). We present an $O(\log n)$-factor approximation algorithm for this problem.

We also consider the prize collecting version, wherein every job also has a penalty associated with it. The feasible solution consists of a subset of the jobs, and a set of resources, to perform the chosen subset of jobs. The goal is to find a feasible solution that minimizes the sum of the costs of the selected resources and the penalties of the jobs that are not selected. We present a constant factor approximation algorithm for this problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, Partial Covering, Interval Graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.199

1 Introduction

We consider the problem of allocating resources to schedule jobs. Each job is specified by its start-time, end-time and its demand requirement. Each resource is specified by its start-time, end-time, the capacity it offers and its associated cost. A feasible solution is a set of resources satisfying the constraint that at any timeslot, the sum of the capacities offered by the resources is at least the demand required by the jobs active at that timeslot, i.e., the selected resources must cover the jobs. The cost of a feasible solution is the sum of costs of the resources picked in the solution. The goal is to pick a feasible solution having minimum cost. We call this the Resource Allocation problem (ResAll).

The above problem is motivated by applications in cloud and grid computing. Consider jobs that require a common resource such as network bandwidth or storage. The resource may be available under different plans; for instance, it is common for network bandwidth to be priced based on the time of the day to account for the network usage patterns during the day. The plans may offer different capacities of the resource at different costs. Moreover,

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 199–210

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

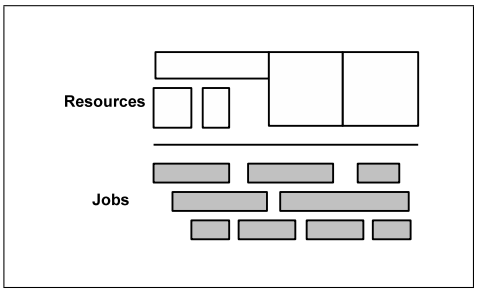


Figure 1 Illustration of the input

It may be possible to lease multiple units of the resource under some plan by paying a cost proportional to the number of units.

Bar-Noy et al. [2] presented a 4-approximation algorithm for the ResAll problem (See Section 4 therein). We consider two variants of the problem. The first variant is the partial covering version. In this problem, the input also specifies a number k and a feasible solution is only required to cover k of the jobs. The second variant is the prize collecting version wherein each job has a penalty associated with it; for every job that is not covered by the solution, the solution incurs an additional cost, equivalent to the penalty corresponding to the job. These variants are motivated by the concept of service level agreements (SLA's), which stipulate that a large fraction of the client's jobs are to be completed. We study these variants for the case where the demands of all the jobs are uniform (say 1 unit) and a solution is allowed to pick multiple copies of a resource by paying proportional cost. We now define our problems formally.

1.1 Problem Definition

We consider the timeline $\mathcal{T}$ to be uniformly divided into discrete intervals ranging from 1 to T. We refer to each integer $1 \leq t \leq T$ as a *timeslot*. The input consists of a set of *jobs* $\mathcal{J}$, and a set of *resources* $\mathcal{R}$.

Each job $j \in \mathcal{J}$ is specified by an interval $I(j) = [s(j), e(j)]$, where $s(j)$ and $e(j)$ are the *start-time* and *end-time* of the job j. We further assume that $s(j)$ and $e(j)$ are integers in the range $[1, T]$ for every job j. While the various jobs may have different intervals associated with them, we consider all the jobs to have *uniform* demand requirement, say 1 unit.

Further, each resource $i \in \mathcal{R}$ is specified by an interval $I(i) = [s(i), e(i)]$, where $s(i)$ and $e(i)$ are the *start-time* and the *end-time* of the resource i; we assume that $s(i)$ and $e(i)$ are integers in the range $[1, T]$. The resource i is also associated with a *capacity* $w(i)$ and a cost $c(i)$; we assume that $w(i)$ is an integer. We interchangeably refer to the resources as *resource intervals*. A typical scenario of such a collection of jobs and resources is shown in Figure 1.

We say that a job j (resource i) is *active* at a timeslot t, if $t \in I(j)$ $(I(i))$; we denote this as $j \sim t$ $(i \sim t)$. In this case, we also say that j (or i) *spans* t.

We define a *profile* $P : \mathcal{T} \to \mathbb{N}$ to be a mapping that assigns an integer value to every timeslot. For two profiles, P_1 and P_2, P_1 is said to *cover* P_2, if $P_1(t) \geq P_2(t)$ for all $t \in \mathcal{T}$. Given a set J of jobs, the profile $P_J(\cdot)$ of J is defined to be the mapping determined by the cumulative demand of the jobs in J, i.e. $P_J(t) = |\{j \in J \ : \ j \sim t\}|$. Similarly, given a multiset R of resources, its profile is: $P_R(t) = \sum_{i \in R \ : \ i \sim t} w(i)$ (taking copies of a resource

into account). We say that R *covers* J if P_R covers P_J. The cost of a multiset of resources R is defined to be the sum of the costs of all the resources (taking copies into account).

We now describe the two versions of the problem.

- PartialResAll: In this problem, the input also specifies a number k (called the *partiality parameter*) that indicates the number of jobs to be covered. A feasible solution is a pair (R, J) where R is a multiset of resources and J is a set of jobs such that R covers J and $|J| \geq k$. The problem is to find a feasible solution of minimum cost.
- PrizeCollectingResAll: In this problem, every job j also has a penalty p_j associated with it. A feasible solution is a pair (R, J) where R is a multiset of resources and J is a set of jobs such that R covers J. The cost of the solution is the sum of the costs of the resources in R and the penalties of the jobs not in J. The problem is to find a feasible solution of minimum cost.

Note that in both the versions, multiple copies of the same resource can be picked by paying the corresponding cost as many times.

1.2 Related Work and Our Results

Our work belongs to the space of *partial* covering problems, which are a natural variant of the corresponding full cover problems. There is a significant body of work that consider such problems in the literature, for instance, see [9, 3, 10, 11, 8].

In the setting where resources and jobs are embodied as intervals, the objective of finding a minimum cost collection of resources that fulfill the jobs is typically called the *full cover* problem. Full cover problems in the interval context have been dealt with earlier, in various earlier works [2, 4, 7]. Partial cover problems in the interval context have been considered earlier in [5].

> **Our Main Result**. We present an $O(\log(n + m))$ approximation for the PartialResAll problem, where n is the number of jobs and m is the number of resources respectively.

The work in existing literature that is closest in spirit to our result is that of Bar-Noy et al.[2], and Chakaravarthy et al.[5]. In [2], the authors consider the full cover version, and present a 4-approximation algorithm. In this case, all the jobs are to be covered, and therefore the demand profile to be covered is fixed. The goal is to find the minimum cost set of resources, for covering this profile. In our setting, we need to cover only k of the jobs. A solution needs to select k jobs to be covered in such a manner that the resources required to cover the resulting demand profile has minimum cost.

In [5], the authors consider a scenario, wherein the timeslots have demands and a solution must satisfy the demand for at least k of the timeslots. In contrast, in our setting, a solution needs to satisfy k *jobs*, wherein each job can span multiple timeslots. A job may not be completely spanned by any resource, and thus may require *multiple* resource intervals for covering it.

We also show a constant factor approximation algorithm for the PrizeCollectingResAll problem, by reducing it to the zero-one version of the ResAll problem. Jain and Vazirani [10] provide a general framework for achieving approximation algorithms for partial covering problems, wherein the prize collecting version is considered. In this framework, under suitable conditions, a constant factor approximation for the prize collecting version implies a constant factor approximation for the partial version as well. However, their result applies only when the prize collecting algorithm has a certain strong property, called the *Lagrangian Multiplier Preserving* (LMP) property. While we are able to achieve a

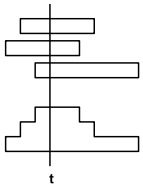


Figure 2 A Mountain M

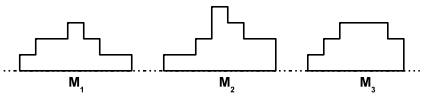


Figure 3 A Mountain Range $\mathcal{M} = \{M_1, M_2, M_3\}$

constant factor approximation for the PrizeCollectingResAll problem, our algorithm does not have the LMP property. Thus, the Jain-Vazirani framework does not apply to our scenario. Due to space constraints, we defer the proof of our algorithm for the PrizeCollectingResAll problem to the full version of the paper [6].

2 Outline of the Main Algorithm

In this section, we outline the proof of our main result:

▶ **Theorem 1.** *There exists an $O(log(n+m))$-approximation algorithm for the* PartialResAll *problem, where n is the number of jobs and m is the number of resources.*

The proof of the above theorem goes via the claim that the input set of jobs can be partitioned into a logarithmic number of *mountain ranges*. A collection of jobs M is called a *mountain* if there exists a timeslot t, such that all the jobs in this collection span the timeslot t; the specified timeslot where the jobs intersect will be called the *peak* timeslot of the mountain (see Figure 2; jobs are shown on the top and the profile is shown below). The justification for this linguistic convention is that if we look at the profile of such a collection of jobs, the profile forms a bitonic sequence, increasing in height until the peak, and then decreasing. The *span* of a mountain is the interval of timeslots where any job in the mountain is active. A collection of jobs $\mathcal{M}$ is called a *mountain range*, if the jobs can be partitioned into a sequence $M_1, M_2, \ldots, M_r$ such that each M_i is a mountain and the spans of any two mountains are non-overlapping (see Figure 3). The decomposition lemma below shows that the input set of jobs can be partitioned into a logarithmic number of mountain ranges. For a job j with start- and end-times $s(j)$ and $e(j)$, let its *length* be $\ell_j = (e(j) - s(j) + 1))$. Let $\ell_{\min}$ be the shortest job length, and $\ell_{\max}$ the longest job length. The proof of the lemma is inspired by the algorithm for the Unsplittable Flow Problem on a line, due to Bansal et al. [1], and it is given in Appendix A.

▶ **Lemma 2.** *The input set of jobs can be partitioned into groups, $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_L$, such that each $\mathcal{M}_i$ is a mountain range and $L \leq 4 \cdot \lceil \log \frac{\ell_{\max}}{\ell_{\min}} \rceil$.*

Theorem 3 (see below) provides a c-approximation algorithm (where c is a constant) for the special case where the input set of jobs form a single mountain range. We now prove Theorem 1, assuming Lemma 2 and Theorem 3.

Proof of Theorem 1. Let $\mathcal{J}$ be the input set of jobs, $\mathcal{R}$ be the input set of resources and k be the partiality parameter. Invoke Lemma 2 on the input set of jobs $\mathcal{J}$ and obtain a partitioning of $\mathcal{J}$ into mountain ranges $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_L$, where $L = 4 \cdot \lceil \log(\ell_{\max}/\ell_{\min}) \rceil$. Theorem 3 provides a c-approximation algorithm $\mathcal{A}$ for the PartialResAll problem wherein the input set of jobs form a single mountain range, where c is some constant. We shall present a (cL)-approximation algorithm for the PartialResAll problem.

For $1 \leq q \leq L$ and $1 \leq \kappa \leq k$, let $\mathcal{A}(q, \kappa)$ denote the cost of the (approximately optimal) solution returned by the algorithm in Theorem 3 with $\mathcal{M}_q$ as the input set of jobs, $\mathcal{R}$ as the input set of resources and κ as the partiality parameter. Similarly, let $\mathrm{OPT}(q, \kappa)$ denote the cost of the optimal solution for covering κ of the jobs in the mountain range $\mathcal{M}_q$. Theorem 3 implies that $\mathcal{A}(q, \kappa) \leq c \cdot \mathrm{OPT}(q, \kappa)$.

The algorithm employs dynamic programming. We maintain a 2-dimensional DP table $\mathrm{DP}[\cdot, \cdot]$. For each $1 \leq q \leq L$ and $1 \leq \kappa \leq k$, the entry $\mathrm{DP}[q, \kappa]$ would store the cost of a (near-optimal) feasible solution covering κ of the jobs from $\mathcal{M}_1 \cup \mathcal{M}_2 \cup \cdots \cup \mathcal{M}_q$. The entries are calculated as follows.

$$\mathrm{DP}[q, \kappa] = \min_{\kappa' \leq \kappa} \{\mathrm{DP}[q-1, \kappa - \kappa'] + \mathcal{A}(q, \kappa')\}.$$

The above recurrence relation considers covering κ' jobs from the mountain $\mathcal{M}_q$, and the remaining $\kappa - \kappa'$ jobs from the mountain ranges $\mathcal{M}_1, \cdots, \mathcal{M}_{q-1}$. Using this dynamic program, we compute a feasible solution to the original problem instance (i.e., covering k jobs from all the mountain ranges $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_L$); the solution would correspond to the entry $\mathrm{DP}[L, k]$. Consider the optimum solution OPT to the original problem instance. Suppose that OPT covers k_q jobs from the mountain range $\mathcal{M}_q$ (for $1 \leq q \leq L$), such that $k_1 + k_2 + \cdots + k_L = k$. Observe that

$$\begin{aligned} \mathrm{DP}[L, k] &\leq \sum_{q=1}^{L} \mathcal{A}(q, k_q) \\ &\leq c \cdot \sum_{q=1}^{L} \mathrm{OPT}(q, k_q), \end{aligned}$$

where the first statement follows from the construction of the dynamic programming table and the second statement follows from the guarantee given by algorithm $\mathcal{A}$. However the maximum of $\mathrm{OPT}(q, k_q)$ (over all q) is a lower bound for OPT (we cannot say anything stronger than this since OPT might use the same resources to cover jobs across multiple subsets $\mathcal{M}_q$). This implies that $\mathrm{DP}[L, k] \leq c \cdot L \cdot \mathrm{OPT}$. This proves the (cL)-approximation ratio.

It is easy to see that L is $O(\log(n+m))$ as argued below. It suffices if we consider the timeslots where some job or resource starts or ends; the other timeslots can be ignored. Such a transformation will not affect the set of feasible solutions. Thus, without loss of generality, we can assume that the number of timeslots $T \leq 2(n+m)$. Therefore, $\ell_{\max} \leq 2(n+m)$ and $\ell_{\min} \geq 1$. Hence, the overall algorithm has an $O(\log(n+m))$ approximation ratio. ◀

▶ **Theorem 3.** *There exists a constant factor approximation algorithm for the special case of the* PartialResAll *problem, wherein the input set of jobs form a single mountain range* $\mathcal{M}$.

The first step in proving the above theorem is to design an algorithm for handling the special case where the input set of jobs form a single mountain. This is accomplished by the following theorem. The proof is given in Section 3.

▶ **Theorem 4.** *There exists an 8-approximation algorithm for the special case of the* PartialResAll *problem wherein the input set of jobs for a single mountain* M.

We now sketch the proof of Theorem 3. Let the input mountain range be $\mathcal{M}$ consisting of mountains $M_1, M_2, \ldots, M_r$. The basic intuition behind the algorithm is to "collapse" each

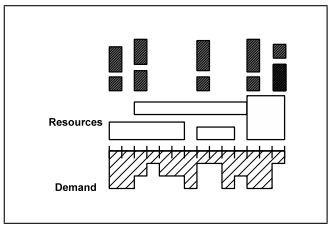


Figure 4 The LSPC problem

mountain M_q into a single timeslot. A resource interval i is said to intersect a mountain M if the interval i and the span of M overlap; the resource i is said to *fully span* the mountain M, if the span of M is contained in the interval i; the resource i is said to be contained in the mountain M, if the interval i is contained in the span of M. It may be possible that for a resource interval i and a mountain M, neither i fully spans M nor is i contained in M. However, at a factor three loss in the approximation ratio, we can transform an input instance into an instance satisfying the following property. The resource intervals in the modified instance can be classified into two categories: (1) *narrow* resources i having the property that the interval i is contained in the span of a specific single mountain M; (2) *wide* resources i having the property that if i intersects any mountain M, then it fully spans the mountain.

The notion of collapsing mountains into timeslots is natural when the input instance consists only of wide resources. This is because we can collapse the mountains $M_1, M_2, \ldots, M_r$ into timeslots $1, 2, \ldots, r$. Furthermore, for each wide resource i, consider the sequence of mountains $M_p, M_{p+1}, \ldots, M_q$ (for some $p \leq q$) that are fully spanned by the resource i; then we represent i by an interval that spans the timeslots $[p, q]$. However, the case of narrow resources is more involved because a narrow resource does not fully span the mountain containing it. Based on the above intuition, we define a problem called the *Long Short Partial Cover* (LSPC). The algorithm for handling a mountain range goes via a reduction to the LSPC problem.

*Problem Definition (*LSPC*):* We are given a demand profile over a range $[1, T]$, which specifies an integral demand d_t at each timeslot $t \in [1, T]$. The input resources are of two types, *short* and *long*. A short resource spans only one timeslot, whereas a long resource can span one or more timeslots. Each resource i has a cost $c(i)$ and a capacity $w(i)$. The input also specifies a *partiality parameter* k. A feasible solution S consists of a multiset of resources S and a coverage profile. A *coverage profile* is a function that assigns an integer k_t for each timeslot t satisfying $k_t \leq d_t$. The solution should have the following properties: (i) $\sum_t k_t \geq k$; (ii) at any timeslot t, the sum of capacities of the resource intervals from S active at t is at least k_t; (iii) for any timeslot t, at most one of the short resources spanning the timeslot t is picked (however, multiple copies of a long resource may be included). The objective is to find a feasible solution having minimum cost. See Figure 4 for an example (in the figure, short resources are shaded).

The advantage with the LSPC problem is that the demands are restricted to single timeslots; in contrast, in the PartialResAll problem, the demands or jobs can span multiple timeslots. Theorem 5 (see below) shows that the LSPC problem can be approximated within a factor of 16. The reduction from the PartialResAll problem restricted to a single mountain range (as in Theorem 3) to the LSPC problem goes by representing each mountain in the input mountain range $\mathcal{M}$ by a single timeslot in the LSPC instance; the

wide resources will correspond to long resources in the LSPC instance. The reduction handles the narrow resources using the short resources; the constraint (iii) in the LSPC problem definition is crucially employed in this process. The reduction from the case of single mountain range to the LSPC problem is deferred to the full version of the paper[6] and a complete proof of Theorem 3 also appears there.

▶ **Theorem 5.** *There exists a* 16*-approximation algorithm for the* LSPC *problem.*

The algorithm claimed in the above theorem is inspired by the work of [5]. In that paper, the authors study a variant of the problem; in that variant, there are only long resources and a solution S must satisfy a set of k timeslots $t_1, t_2, \ldots, t_k \in [1, T]$, where a timeslot t is satisfied, if the sum of capacities of the resources in S active at t is at least the demand d_t; a solution is allowed to pick multiple copies of any resource (both long and short). The LSPC problem differs in two ways: first, a solution can satisfy the demand at a timeslot partially and secondly, only one copy of a short resource can be picked. These two differences give rise to complications and as a result, our algorithm is more involved. The algorithm is provided in Section 4.

3 A Single Mountain: Proof of Theorem 4

In this section, we give an 8-factor approximation algorithm for the case of the PartialResAll problem, where the input jobs form a single mountain.

The basic intuition is as follows. Given the structure of the jobs, we will show that there is a *near-optimal* feasible solution that exhibits a nice property: the jobs discarded from the solution are extremal either in their start-times or their end-times.

▶ **Lemma 6.** *Consider the* PartialResAll *problem for a single mountain. Let* $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ *be the input set of jobs. Let* $S = (R_S, J_S)$ *be a feasible solution such that* R_S *covers the set of jobs* J_S *with* $|J_S| = k$*. Let* C_S *denote its cost. Let* $L =< l_1, l_2, \ldots, l_n >$ *denote the jobs in increasing order of their start-times. Similarly, let* $R =< r_1, r_2, \ldots, r_n >$ *denote the jobs in decreasing order of their end-times. Then, there exists a feasible solution* $X = (R_X, J_X)$ *having cost at most* $2 \cdot C_S$ *such that*

$$\mathcal{J} \setminus J_X = \{l_i : i \leq q_1\} \cup \{r_i : i \leq q_2\} \tag{1}$$

for some $q_1, q_2 \geq 0$ *where* $|\mathcal{J} \setminus J_X| = n - k$*.*

Proof. We give a constructive proof to determine the sets J_X and R_X.

We initialize the set $J_X = \mathcal{J}$. At the end of the algorithm, the set J_X will be the desired set of jobs covered by the solution. The idea is to remove the jobs that extend most to the right or the left from the consideration of J_X. The most critical aspect of the construction is to ensure that whenever we exclude any job from consideration of J_X that is already part of J_S, we do so in pairs of the leftmost and rightmost extending jobs of J_S that are still remaining in J_X. We terminate this process when the size of J_X equals the size of J_S, i.e., k. We also initialize the set $U = \phi$. At the end of the algorithm, this set will contain the set of jobs removed from $\mathcal{J}$ that belonged to J_S while constructing J_X.

We now describe the construction of J_X formally. We maintain two pointers *l-ptr* and *r-ptr*; *l-ptr* indexes the jobs in the sequence L and *r-ptr* indexes the jobs in the sequence R. We keep incrementing the pointer *l-ptr* and removing the corresponding job from J_X (if it has not already been removed) until either the size of J_X reaches k or we encounter a job

(say *l-job*) in J_X that belongs to J_S; we do not yet remove the job *l-job*. We now switch to the pointer *r-ptr* and start incrementing it and removing the corresponding job from J_X (if it has not already been removed) until either the size of J_X reaches k or we encounter a job (say *r-job*) in J_X that belongs to J_S; we do not yet remove the job *r-job*. If the size of J_X reaches k, we have the required set J_X.

Now suppose that $|J_X| \neq k$. Note that both *l-ptr* and *r-ptr* are pointing to jobs in J_S. Let *l-job* and *r-job* be the jobs pointed to by *l-ptr* and *r-ptr* respectively (note that these two jobs may be same).

We shall remove one or both of *l-job* and *r-job* from J_X and put them in U. We classify these jobs into three categories: *single*, *paired* and *artificially paired*.

Suppose that $|J_X| \geq k+2$. In this case, we have to delete at least 2 more jobs; so we delete both *l-job* and *r-job* and add them to U as *paired* jobs. In case *l-job* and *r-job* are the same job, we just delete this job and add it to U as a *single* job. We also increment the *l-ptr* and *r-ptr* pointers to the next job indices in their respective sequence. We then repeat the same process again, searching for another pair of jobs.

Suppose that $|J_X| = k+1$. In case *l-job* and *r-job* are the same job, we just delete this job and get the required set J_X of size k; We add this job to the set U as a *single* job. On the other hand, if *l-job* and *r-job* are different jobs, we remove *l-job* from J_X and add it to U as *artificially paired* with its pair as the job *r-job* ; note that we do not remove *r-job* from J_X.

This procedure gives us the required set J_X. We now construct R_X by simply doubling the resources of R_S; meaning, that for each resource in R_S, we take twice the number of copies in R_X. Clearly $C_X = 2 \cdot C_S$. It remains to argue that R_X covers J_X. For this, note that $U = J_S - J_X$ and hence $|U| = |J_X - J_S|$ (because $|J_X| = |J_S| = k$). We create an arbitrary bijection $f : U \to J_X - J_S$. Note that J_X can be obtained from J_S by deleting the jobs in U and adding the jobs of $J_X - J_S$.

We now make an important observation:

▶ Observation 7. For any *paired* jobs or *artificially paired* jobs j_1, j_2 added to U, all the jobs in J_X are contained within the span of this pair, i.e., for any j in J_X, $s_j \geq \min\{s(j_1), s(j_2)\}$ and $e_j \leq \max\{e(j_1), e(j_2)\}$. Similarly for any *single* job j_1 added to U, all jobs in J_X are contained in the span of j_1.

For every *paired* jobs, j_1, j_2, Observation 7 implies that taking 2 copies of the resources covering $\{j_1, j_2\}$ suffices to cover $\{f(j_1), f(j_2)\}$. Similarly, for every *single* job j, the resources covering $\{j\}$ suffice to cover $\{f(j)\}$. Lastly for every *artificially paired* jobs j_1, j_2 where $j_1 \in U$ and $j_2 \notin U$, taking 2 copies of the resources covering $\{j_1, j_2\}$ suffices to cover $\{f(j_1), j_2\}$.

Hence the set R_X obtained by doubling the resources R_S (that cover J_S) suffices to cover the jobs in J_X. ◀

Recall that Bar-Noy et al. [2] presented a 4-approximation algorithm for the ResAll problem (full cover version). Our algorithm for handling a single mountain works as follows. Given a mountain consisting of the collection of jobs $\mathcal{J}$ and the number k, do the following for all possible pairs of numbers (q_1, q_2) such that the set J_X defined as per Equation 1 in Lemma 6 has size k. For the collection of jobs J_X, consider the issue of selecting a minimum cost set of resources to cover these jobs; note that this is a full cover problem. Thus, the 4-approximation of [2] can be applied here. Finally, we output the best solution across all choices of (q_1, q_2). Lemma 6 shows that this is an 8-factor approximation to the PartialResAll problem for a single mountain.

4 LSPC Problem: Proof of Theorem 5

Here, we present a 16-approximation algorithm for the LSPC problem.

We extend the notion of profiles and coverage to ranges contained within $[1, T]$. Let $[a, b]$ contained in $[1, T]$ be a timerange. By a profile over $[a, b]$, we mean a function Q that assigns a value $Q(t)$ to each timeslot $t \in [a, b]$. A profile Q defined over a range $[a, b]$ is said to be *good*, if for all timeslots $t \in [a, b]$, $Q(t) \leq d_t$ (where d_t is the input demand at t). In the remainder of the discussion, we shall only consider good profiles and so, we shall simply write "profile" to mean a "good profile". The *measure* of Q is defined to be the sum $\sum_{t\in[a,b]} Q(t)$.

Let S be a multiset of resources and let Q be a profile over a range of timeslots $[a, b]$. We say that S is *good*, if it includes at most one short resource active at any timeslot t. We say that S covers the profile Q, if for any timeslot $t \in [a, b]$, the sum of capacities of resources in S active at t is at least $Q(t)$. Notice that S is a feasible solution to the input problem instance, if there exists a profile Q over the entire range $[1, T]$ such that Q has measure k and S is a cover for Q. For a timeslot $t \in [1, T]$, let $Q_S^{\text{sh}}(t)$ denote the capacity of the unique short resource from S active at t, if one exists; otherwise, $Q_S^{\text{sh}}(t) = 0$.

Let S be a good multiset of resources and let Q be a profile over a range of timeslots $[a, b]$. For a long resource $i \in S$, let $f_S(i)$ denote the number of copies of i included in S. The multiset S is said to be a *single long resource assignment cover* (SLRA cover) for Q, if for any timeslot $t \in [a, b]$, there exists a long resource $i \in S$ such that $w(i)f_S(i) \geq Q(t) - Q_S^{\text{sh}}(t)$ (intuitively, the resource i can cover the residual demand by itself, even though other long resources in S may be active at t).

We say that a good multiset of resources S is an *SLRA solution* to the input LSPC problem instance, if there exists a profile Q over the range $[1, T]$ having measure k such that S is an SLRA cover for Q. The lemma below shows that near-optimal SLRA solutions exist.

▶ **Lemma 8.** *Consider the input instance of the* LSPC *problem. There exists an SLRA solution having cost at most 16 times the cost of the optimal solution.*

The lemma follows from a similar result proved in [5] and the proof is deferred to the full version of the paper[6]. Surprisingly, we can find the *optimum* SLRA solution S^* in polynomial time, as shown in Theorem 9 below. Lemma 8 and Theorem 9 imply that S^* is a 16-factor approximation to the optimum solution. This completes the proof of Theorem 5.

▶ **Theorem 9.** *The optimum SLRA solution S^* can be found in time polynomial in the number of resources, number of timeslots and H, where $H = \max_{t\in[1,T]} d_t$.*

The rest of the section is devoted to proving Theorem 9. The algorithm goes via dynamic programming. The following notation is useful in our discussion.

- Let S be a good set consisting of only short resources, and let $[a, b]$ be a range. For a profile Q defined over $[a, b]$, S is said to be an *h-free cover* for Q, if for any $t \in [a, b]$, $Q_S^{\text{sh}}(t) \geq Q(t) - h$. The set S is said to be an *h-free q-cover* for $[a, b]$, if there exists a profile Q over $[a, b]$ such that Q has measure q and S is a h-free cover for Q.
- Let S be a good multiset of resources and let $[a, b]$ be a range. For a profile Q defined over $[a, b]$, the multiset S is said to be an *h-free SLRA cover* for Q, if for any timeslot $t \in [a, b]$ satisfying $Q(t) - Q_S^{\text{sh}}(t) > h$, there exists a long resource $i \in S$ such that $w(i)f_S(i) \geq Q(t) - Q_S^{\text{sh}}(t)$. For an integer q, we say S is an *h-free SLRA q-cover* for the range $[a, b]$, if there exists a profile Q over $[a, b]$ such that Q has measure q and S is a h-free SLRA cover for Q.

Intuitively, h denotes the demand covered by long resources already selected (and their cost accounted for) in the previous stages of the algorithm; thus, timeslots whose residual demand is at most h can be ignored. The notion of "h-freeness" captures this concept.

We shall first argue that any h-free SLRA cover S for a profile Q over a timerange $[a, b]$ exhibits certain interesting decomposition property. Intuitively, in most cases, the timeline can be partitioned into two parts (left and right), and S can be partitioned into two parts S_1 and S_2 such that S_1 can cover the left timerange and S_2 can cover the right timerange (even though resources in S_1 may be active in the right timerange and those in S_2 may be active in the left timerange). In the cases where the above decomposition is not possible, there exists a long resource spanning almost the entire range. The lemma is similar to a result proved in [5] (see Lemma 4 therein). The proof is deferred to the full version of the paper[6].

▶ **Lemma 10.** *Let $[a, b]$ be any timerange, Q be a profile over $[a, b]$ and let h be an integer. Let S be a good set of resources providing an h-free SLRA-cover for Q. Then, one of the following three cases holds:*

- *The set of short resources in S form a h-free cover for Q.*
- *Time-cut: There exists a timeslot $a \leq t^* \leq b-1$ and a partitioning of S into S_1 and S_2 such that S_1 is an h-free SLRA-cover for Q_1 and S_2 is an h-free SLRA-cover for Q_2, where Q_1 and Q_2 profiles obtained by restricting Q to $[a, t^*]$ and $[t^*+1, b]$, respectively.*
- *Interval-cut: There exists a long resource $i^* \in S$ such that the set of short resources in S forms a h-free cover for both Q_1 and Q_2, where Q_1 and Q_2 are the profiles obtained by restricting Q to $[a, s(i^*)-1]$ and $[e(i^*)+1, b]$ respectively.*

We now discuss our dynamic programming algorithm. Let $H = \max_{t\in[1,T]} d_t$ be the maximum of the input demands. The algorithm maintains a table M with an entry for each triple $\langle [a, b], q, h\rangle$, where $[a, b] \subseteq [1, T]$, $0 \leq q \leq k$ and $0 \leq h \leq H$. The entry $M([a, b], q, h)$ stores the cost of the optimum h-free SLRA q-cover for the range $[a, b]$; if no solution exists, then $M([a, b], q, h)$ will be ∞. Our algorithm outputs the solution corresponding to the entry $M([1, T], k, 0)$; notice that this is optimum SLRA solution S^*.

In order to compute the table M, we need an auxiliary table A. For a triple $[a, b]$, q and h, let $A([a, b], q, h)$ be the optimum h-free q-cover for $[a, b]$, (using only the short resources); if no solution exists $A([a, b], q, h)$ is said to be ∞. We first describe how to compute the auxiliary table A. For a triple consisting of $t \in [1, T]$, $q \leq k$ and $h \leq H$, define $\gamma(t, q, h)$ as follows. If $q > d_t$, set $\gamma(t, q, h) = \infty$. Consider the case where $q \leq d_t$. If $q \leq h$, set $\gamma(t, q, h) = 0$. Otherwise, let i be the minimum cost short resource active at t such that $w(i) \geq q - h$; set $\gamma(t, q, h) = c(i)$; if no such short resource exists, set $\gamma(q, t, h) = \infty$.

Then, for a triple $\langle [a, b], q, h\rangle$, the entry $A([a, b], q, h)$ is governed by the following recurrence relation. Of the demand q that need to be covered, the optimum solution may cover a demand q_1 from the timeslot t, and a demand $q - q_1$ from the range $[a, b-1]$. We try all possible values for q_1 and choose the best:

$$A([a, b], q, h) = \min_{q_1 \leq \min\{q, d_b\}} A([a, b-1], q - q_1, h) + \gamma(b, q_1, h).$$

It is not difficult to verify the correctness of the above recurrence relation.

We now describe how to compute the table M. Based on the decomposition lemma (Lemma 10), we can develop a recurrence relation for a triple $[a, b]$, q and h. We compute $M([a, b], q, h)$ as the minimum over three quantities E_1, E_2 and E_3 corresponding to the

$$
\begin{aligned}
E_1 &= A([a,b],q,h).\\
E_2 &= \min_{\substack{t\in[a,b-1]\\ q_1\le q}} M([a,t],q_1,h)+M([t+1,b],q-q_1,h).\\
E_3 &= \min_{\substack{(i\in\mathcal{L},\alpha\le H)\ :\ \alpha w(i)>h\\ q_1,q_2,q_3\ :\ q_1+q_2+q_3=q}} \begin{pmatrix} \alpha\cdot c(i)\\ +A([a,s(i)-1],q_1,h)\\ +M([s(i),e(i)],q_2,\alpha w(i))\\ +A([e(i)+1,b],q_3,h)\end{pmatrix}
\end{aligned}
$$

Figure 5 Recurrence relation for M

three cases of the lemma. Intuitive description of the three quantities is given below and precise formulas are provided in Figure 5. In the figure, $\mathcal{L}$ is the set of all long resources[1].

- *Case 1:* No long resource is used and so, we just use the corresponding entry of the table A.
- *Case 2:* There exists a time-cut t^*. We consider all possible values of t^*. For each possible value, we try all possible ways in which q can be divided between the left and right ranges.
- *Case 3:* There exists a long resource i^* such that the timeranges to the left of and to the right of i^* can be covered solely by short resources. We consider all the long resources i and also the number of copies α to be picked. Once α copies of i are picked, i can cover all timeslots with residual demand at most $\alpha w(i)$ in an SLRA fashion, and so the subsequent recursive calls can ignore these timeslots. Hence, this value is passed to the recursive call. We also consider different ways in which q can be split into three parts - left, middle and right. The left and right parts will be covered by the solely short resources and the middle part will use both short and long resources. Since we pick α copies of i, a cost of $\alpha c(i)$ is added.

We set $M([a,b],q,h)=\min\{E_1,E_2,E_3\}$. For the base case: for any $[a,b]$, if $q=0$ or $h=H$, then the entry is set to zero.

We now describe the order in which the entries of the table are filled. Define a partial order $\prec$ as below. For pair of triples $z=([a,b],q,h)$ and $z'=([a',b'],q',h')$, we say that $z\prec z'$, if one of the following properties is true: (i)$[a',b']\subseteq[a,b]$; (ii) $[a,b]=[a',b']$ and $q<q'$; (iii) $[a,b]=[a',b']$, $q=q'$ and $h>h'$. Construct a directed acyclic graph (DAG) G where the triples are the vertices and an edge is drawn from a triple z to a triple z', if $z\prec z'$. Let π be a topological ordering of the vertices in G. We fill the entries of the table M in the order of appearance in π. Notice that the computation for any triple z only refers to triples appearing earlier than z in π.

Using Lemma 10, we can argue that the above recurrence relation correctly computes all the entries of M.

References

1 N. Bansal, Z. Friggstad, R. Khandekar, and M. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA*, 2009.

[1] The input demands d_t are used in computing the table $A(\cdot,\cdot,\cdot)$

2 A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

3 R. Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. *J. Algorithms*, 39(2):137–144, 2001.

4 R. Bhatia, J. Chuzhoy, A. Freund, and J. Naor. Algorithmic aspects of bandwidth trading. *ACM Transactions on Algorithms*, 3(1), 2007.

5 V. Chakaravarthy, A. Kumar, S. Roy, and Y. Sabharwal. Resource allocation for covering time varying demands. In *ESA*, 2011.

6 V. Chakaravarthy, A. Pal, S. Roy, and Y. Sabharwal. Scheduling resources for executing a partial set of jobs. *CoRR*, abs/1210.2906, 2012.

7 D. Chakrabarty, E. Grant, and J. Könemann. On column-restricted and priority covering integer programs. In *IPCO*, pages 355–368, 2010.

8 R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

9 N. Garg. Saving an ϵ: a 2-approximation for the k-MST problem in graphs. In *STOC*, 2005.

10 K. Jain and V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

11 J. Könemann, O. Parekh, and D. Segev. A unified approach to approximating partial covering problems. *Algorithmica*, 59(4), 2011.

A Proof of Lemma 2

We first categorize the jobs according to their lengths into r categories C_1, C_2, $\cdots, C_r$, where $r = \lceil \log \frac{\ell_{\max}}{\ell_{\min}} \rceil$. The category C_i consists of all the jobs with lengths in the range $[2^{i-1}\ell_{\min}, 2^i \ell_{\min})$. Thus all the jobs in any single category have comparable lengths: any two jobs j_1 and j_2 in the category satisfy $\ell_1 < 2\ell_2$, where ℓ_1 and ℓ_2 are the lengths of j_1 and j_2 respectively.

Consider any category C and let the lengths of the jobs in C lie in the range $[\alpha, 2\alpha)$. We claim that the category C can be partitioned into 4 groups G_0, G_1, G_2, G_3, such that each G_i is a mountain range. To see this, partition the set of jobs C into classes $H_1, H_2, \ldots, H_q, \ldots$ where H_q consists of the jobs active at timeslot $q \cdot \alpha$. Note that every job belongs to some class since all the jobs have length at least α; if a job belongs to more than one class, assign it to any one class arbitrarily. Clearly each class H_q forms a mountain. For $0 \leq i \leq 3$, let G_i be the union of the classes H_q satisfying $q \equiv i \mod 4$. Since each job has length at most 2α, each G_i is a mountain range. Thus, we get a decomposition of the input jobs into $4r$ mountain ranges. ◀

Visibly Rational Expressions

Laura Bozzelli[1] and César Sánchez[2]

1 **Technical University of Madrid (UPM), Madrid, Spain**
`laura.bozzelli@fi.upm.es`
2 **IMDEA Software Institute, Madrid, Spain &**
Institute for Applied Physics, CSIC, Madrid, Spain
`cesar.sanchez@imdea.org`

Abstract

Regular Expressions (RE) are an algebraic formalism for expressing regular languages, widely used in string search and as a specification language in verification. In this paper we introduce and investigate Visibly Rational Expressions (VRE), an extension of RE for the well-known class of Visibly Pushdown Languages (VPL). We show that VRE capture the class of VPL. Moreover, we identify an *equally expressive* fragment of VRE which admits a quadratic time compositional translation into the automata acceptors of VPL. We also prove that, for this fragment, universality, inclusion and language equivalence are EXPTIME-complete. Finally, we provide an extension of VRE for VPL over infinite words.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Visibly Pushdown Languages, Context-free specifications, Regular expressions, Algebraic characterization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.211

1 Introduction

Visibly Pushdown Languages (VPL), introduced by Alur et al. [4, 5], represent a robust and widely investigated subclass of context-free languages which includes strictly the class of regular languages. A VPL consists of *nested words*, that is words over an alphabet (pushdown alphabet) which is partitioned into three disjoint sets of calls, returns, and internal symbols. This partition induces a nested hierarchical structure in a given word obtained by associating to each call the corresponding matching return (if any) in a well-nested manner. VPL are accepted by Visibly Pushdown Automata (VPA), a subclass of pushdown automata which push onto the stack only when a call is read, pops the stack only at returns, and do not use the stack on reading internal symbols. Hence, the input controls the kind of operations permissible on the stack, and thus the stack depth at every position [4]. This restriction makes the class of VPL very similar in tractability and robustness to that of regular languages. In particular, VPL are closed under intersection, union, complementation, renaming, concatenation and Kleene closure [4]. Moreover, VPA (over finite words) are determinizable, and decision problems like universality, equivalence and inclusion – which are undecidable for context-free languages – become EXPTIME-complete for VPL. Furthermore, various alternative and constructive characterizations of VPL have been given in terms of operational and descriptive formalisms: logical characterizations by standard MSO over nested words extended with a binary matching-predicate (MSO_μ) [4] or by fixpoint logics [10], a context-free grammar based characterization [4], alternating automata based characterizations [6, 10], and a congruence-based characterization [3].

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 211–223

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

The theory of VPL has relevant applications in the formal verification and synthesis of sequential recursive programs with finite data types modeled by pushdown systems [7, 5, 2, 18]. Runs in these programs can be seen as nested words capturing the nested calling structure, where the call to and return from procedures capture the nesting. Additionally, VPL have applications in the streaming processing of semi-structured data, such as XML documents, where each open-tag is matched with a closing-tag in a well-nested manner [20, 16, 19, 1, 21]. Examples include type-checking (validation) and dynamic typing of XML documents against schema specifications [16], and evaluation of MSO_μ queries on streaming XML documents [19].

Contribution. Regular Expressions [15, 14] (RE) are an algebraic formalism for describing regular languages. RE are widely adopted as a descriptive specification language, for example in string search [22], and for extensions of temporal logics for hardware model checking [13, 17]. In this paper, we introduce and investigate a similar algebraic formalism for the class of VPL, that we call *Visibly Regular Expressions* (VRE). VRE extend RE by adding two novel non-regular operators which are parameterized by an internal action: (1) the binary *Minimally Well-Matched Substitution* operator (M-substitution for short), which allows to substitute occurrences of the designated internal action by *minimally well-matched* (*MWM*) words;[1] (2) and the unary *Strict Mimimally Well-Matched Closure* operator (S-closure for short), which corresponds to the (unbounded) iteration of the M-substitution operation. We also consider a third operator which can be expressed in terms of M-substitution and S-closure. The class of *pure* VRE is obtained by disallowing the explicit use of this operator. Intuitively, M-substitution and S-closure, when applied to languages $\mathcal{L}$ of *MWM* words, correspond to classical tree language concatenation and Kleene closure applied to the tree language encoding of the (nested) word languages $\mathcal{L}$ (in accordance with the standard encoding of *MWM* words by ordered unranked finite trees [1]).

Our results are as follows. First, we establish that VRE capture exactly the class of VPL. Like the classical Kleene theorem [15], the translation from automata (VPA) to expressions (VRE) involves a singly exponential blow-up. For the converse direction (from VRE to VPA), the proposed construction requires again single exponential time (it is an open question if this exponential blow-up can be avoided). On the other hand, we show that pure VRE (which are equivalent to unrestricted VRE) can be compositionally converted in quadratic time into equivalent VPA. The key of this translation is given by a novel subclass of VPA. Next, we prove that universality, inclusion, and language equivalence for pure VRE are EXPTIME-complete. Finally, we also provide an algebraic characterization of VPL over infinite words.

A potential application of our algebraic formalism is as a schema specification language for semi-structured data such as XML documents. In fact, usually, XML schema specifications are context-free grammars and their derivation trees give the tree representation of the associated sets of XML documents. So, these specifications are typically compiled into tree automata. However, it has been shown [16, 19, 1, 21] that VPA are often more natural (and sometime exponentially more succinct) than tree automata, and moreover preferable in the streaming processing of XML documents.

Related Work. Another algebraic characterization of VPL has been given in [8], where regular expressions are extended with an infinite family of operators, which are implicit least fix-points. In fact, these operators encode in a linear way a subclass of context-free grammars. In comparison, we introduce just two operators which make our formalism

[1] *MWM* words are words whose first symbol is a call and whose last symbol is the matching return.

really more lightweight and intuitive to use. In [12] a characterization of VPL is given by morphisms to suitable algebraic structures. Moreover, [12] introduces an extension of the Kleene-closure free fragment of regular expressions obtained by adding a variant of our substitution operator, and shows that this extension captures exactly the first-order fragment of MSO_μ over well-matched words.

2 Visibly pushdown languages

In this section, we recall the class of visibly pushdown automata and visibly pushdown languages [4].

Pushdown Alphabet. A *pushdown alphabet* is a tuple $\widetilde{\Sigma} = \langle \Sigma_{call}, \Sigma_{ret}, \Sigma_{int} \rangle$ consisting of three disjoint finite alphabets: Σ_{call} is a finite set of *calls*, Σ_{ret} is a finite set of *returns*, and Σ_{int} is a finite set of internal actions. For any such $\widetilde{\Sigma}$, the *support* of $\widetilde{\Sigma}$ is $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$. We will use $c, c_1, c_i, \ldots$ for elements of Σ_{call}, $r, r_1, r_i, \ldots$ for elements of Σ_{ret}, $\Box, \Box_1, \Box_j, \ldots$ for elements of Σ_{int}, and $\sigma, \sigma_1, \sigma_i, \ldots$ for arbitrary elements of Σ.

Visibly Pushdown Automata and Visibly Pushdown Languages. A *Nondeterministic Visibly Pushdown Automaton on finite words* (NVPA) [4] *over* $\widetilde{\Sigma} = \langle \Sigma_{call}, \Sigma_{ret}, \Sigma_{int} \rangle$ is a tuple $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F \rangle$, where Q is a finite set of (control) states, $q_{in} \in Q$ is the initial state, Γ is a finite stack alphabet, $\Delta \subseteq (Q \times \Sigma_{call} \times Q \times \Gamma) \cup (Q \times \Sigma_{ret} \times (\Gamma \cup \{\bot\}) \times Q) \cup (Q \times \Sigma_{int} \times Q)$ is a transition relation (where $\bot \notin \Gamma$ is the special *stack bottom symbol*), and $F \subseteq Q$ is a set of accepting states. A transition of the form $(q, c, q', \gamma) \in Q \times \Sigma_{call} \times Q \times \Gamma$ is a push transition, where on reading the call c, the symbol $\gamma \neq \bot$ is pushed onto the stack and the control changes from q to q'. A transition of the form $(q, r, \gamma, q') \in Q \times \Sigma_{ret} \times (\Gamma \cup \{\bot\}) \times Q$ is a pop transition, where on reading the return r, γ is read from the top of the stack and popped, and the control changes from q to q' (if the top of the stack is $\bot$, then it is read but not popped). Finally, on reading an internal action $\Box$, $\mathcal{P}$ can choose only transitions of the form $(q, \Box, q')$ which do not use the stack.

A configuration of $\mathcal{P}$ is a pair (q, β), where $q \in Q$ and $\beta \in \Gamma^* \cdot \{\bot\}$ is a stack content. A run π of $\mathcal{P}$ over a finite word $\sigma_1 \ldots \sigma_{n-1} \in \Sigma^*$ is a finite sequence of the form $\pi = (q_1, \beta_1) \xrightarrow{\sigma_1} (q_2, \beta_2) \ldots \xrightarrow{\sigma_{n-1}} (q_n, \beta_n)$ such that (q_i, β_i) is a configuration for all $1 \leq i \leq n$, and the following holds for all $1 \leq i \leq n-1$:

Push If σ_i is a call, then for some $\gamma \in \Gamma$, $(q_i, \sigma_i, q_{i+1}, \gamma) \in \Delta$ and $\beta_{i+1} = \gamma \cdot \beta_i$.

Pop If σ_i is a return, then for some $\gamma \in \Gamma \cup \{\bot\}$, $(q_i, \sigma_i, \gamma, q_{i+1}) \in \Delta$, and *either* $\gamma \neq \bot$ and $\beta_i = \gamma \cdot \beta_{i+1}$, *or* $\gamma = \bot$ and $\beta_i = \beta_{i+1} = \bot$.

Internal If σ_i is an internal action, then $(q_i, \sigma_i, q_{i+1}) \in \Delta$ and $\beta_{i+1} = \beta_i$.

For all $1 \leq i \leq j \leq n$, the subsequence of π given by $\pi_{ij} = (q_i, \beta_i) \xrightarrow{\sigma_i} \ldots \xrightarrow{\sigma_{j-1}} (q_j, \beta_j)$ is called *subrun of* π (note that π_{ij} is a run over $\sigma_i \ldots \sigma_{j-1}$). The run π is *initialized* if $q_1 = q_{in}$ and $\beta_1 = \bot$. Moreover, the run π is *accepting* if the last state is accepting, that is, if $q_n \in F$. For two configurations (q, β) and (q', β') and a finite word $w \in \Sigma^*$, we write $(q, \beta) \xrightarrow{w} (q', \beta')$ to mean that there is a run of $\mathcal{P}$ over w starting at (q, β) and leading to (q', β'). The language of $\mathcal{P}$, $\mathcal{L}(\mathcal{P})$, is the set of finite words $w \in \Sigma^*$ such that there is an initialized accepting run of $\mathcal{P}$ on w. A language of finite words $\mathcal{L} \subseteq \Sigma^*$ is a *visibly pushdown language* (VPL) *with respect to* $\widetilde{\Sigma}$ if there is a NVPA $\mathcal{P}$ over $\widetilde{\Sigma}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{P})$.

We also consider *Visibly Pushdown Automata on infinite words* (ω-NVPA). Formally, a Büchi ω-NVPA *over* $\widetilde{\Sigma}$ [4] is a tuple $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F \rangle$, where $Q, q_{in}, \Gamma, \Delta$, and F are defined as for NVPA over $\widetilde{\Sigma}$. A run π over an infinite word $\sigma_1 \sigma_2 \ldots \in \Sigma^\omega$ is an infinite sequence $\pi = (q_1, \beta_1) \xrightarrow{\sigma_1} (q_2, \beta_2) \ldots$ that is defined using the natural extension of the defin-

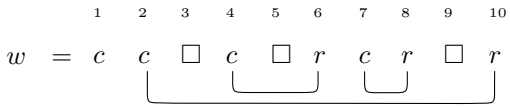
ition of runs on finite words. The run is *accepting* if for infinitely many $i \geq 1$, $q_i \in F$. The notions of initialized run and (finite) subrun are defined as for NVPA. The ω-language of $\mathcal{P}$, $\mathcal{L}(\mathcal{P})$, is the set of infinite words $w \in \Sigma^\omega$ such that there is an initialized accepting run of $\mathcal{P}$ on w. An ω-language $\mathcal{L} \subseteq \Sigma^\omega$ is an *ω-visibly pushdown language* (ω-VPL) *with respect to* $\widetilde{\Sigma}$ if there is a Büchi ω-NVPA $\mathcal{P}$ over $\widetilde{\Sigma}$ such that $\mathcal{L} = \mathcal{L}(P)$.

Matched calls and returns. Fix a pushdown alphabet $\widetilde{\Sigma} = \langle \Sigma_{call}, \Sigma_{ret}, \Sigma_{int} \rangle$. For a finite or infinite word w over Σ, $|w|$ is the length of w (we set $|w| = \omega$ if w is infinite). For all $1 \leq i \leq |w|$, $w(i)$ is the i^{th} symbol of w. A position $1 \leq i \leq |w|$ of w is a call (resp., return, internal) position if $w(i) \in \Sigma_{call}$ (resp., $w(i) \in \Sigma_{ret}$, $w(i) \in \Sigma_{int}$).

The set $WM(\widetilde{\Sigma})$ of *well-matched words* is the subset of Σ^* inductively defined as follows: (i) $\epsilon \in WM(\widetilde{\Sigma})$, where ϵ denotes the empty string, (ii) $\Box \cdot w \in WM(\widetilde{\Sigma})$ if $\Box \in \Sigma_{int}$ and $w \in WM(\widetilde{\Sigma})$, and (iii) $c \cdot w \cdot r \cdot w' \in WM(\widetilde{\Sigma})$ if $c \in \Sigma_{call}$, $r \in \Sigma_{ret}$, and $w, w' \in WM(\widetilde{\Sigma})$.

Let i be a call position of a word w. If there is $j > i$ such that j is a return position of w and $w(i+1) \dots w(j-1)$ is a well-matched word (note that j is uniquely determined if it exists), we say that j is the *matching return* of i along w, and i is the *matching call* of j along w. The set $MWM(\widetilde{\Sigma})$ of *minimally well-matched words* is the set of well-matched words of the form $c \cdot w \cdot r$ such that c is a call, r is a return, and w is well-matched (note that r corresponds to the matching return of c). For a language $\mathcal{L} \subseteq \Sigma^*$, we define $MWM(\mathcal{L}) \stackrel{\text{def}}{=} \mathcal{L} \cap MWM(\widetilde{\Sigma})$, that is the set of words in $\mathcal{L}$ which are minimally well-matched.

▶ **Example 1.** Let $\Sigma_{call} = \{c\}$, $\Sigma_{ret} = \{r\}$, and $\Sigma_{int} = \{\Box\}$. Consider the word w below. The word w is not well-matched, in particular, the call at position 1 has no matching return in w. Moreover, note that the subword $w[2] \dots w[10]$ is minimally well-matched.



3 Visibly Rational Expressions (VRE)

In this section, we introduce and investigate the class of *Visibly Rational Expressions* (VRE), an extension of regular expressions obtained by adding two novel non-regular operators: the binary M-substitution operator and the unary S-closure operator. First, we define the corresponding operations on languages of finite words and show that VPL are effectively closed under these operations. Then, in Subsection 3.1, we introduce VRE and establish their effective language equivalence to the class of NVPA.

Let us fix a pushdown alphabet $\widetilde{\Sigma} = \langle \Sigma_{call}, \Sigma_{ret}, \Sigma_{int} \rangle$. For two languages $\mathcal{L}, \mathcal{L}' \subseteq \Sigma^*$ of finite words on Σ, we use $\mathcal{L} \cdot \mathcal{L}'$ to denote the standard concatenation of $\mathcal{L}$ and $\mathcal{L}'$, and $\mathcal{L}^*$ to denote the standard Kleene closure of $\mathcal{L}$.

▶ **Definition 2** (M-substitution). *Let $w \in \Sigma^*$, $\Box \in \Sigma_{int}$, and $\mathcal{L} \subseteq \Sigma^*$. The M-substitution of $\Box$ by $\mathcal{L}$ in w*, denoted by $w \curvearrowleft_\Box \mathcal{L}$, is the language of finite words over Σ obtained by replacing occurrences of $\Box$ in w by minimally well-matched words in $\mathcal{L}$. Formally, $w \curvearrowleft_\Box \mathcal{L}$ is inductively defined as follows:

- $\varepsilon \curvearrowleft_\Box \mathcal{L} \stackrel{\text{def}}{=} \{\varepsilon\}$;
- $(\Box \cdot w') \curvearrowleft_\Box \mathcal{L} \stackrel{\text{def}}{=} \big(MWM(\mathcal{L}) \cdot (w' \curvearrowleft_\Box \mathcal{L})\big) \cup \big((\{\Box\} \cap \mathcal{L}) \cdot (w' \curvearrowleft_\Box \mathcal{L})\big)$
- $(\sigma \cdot w') \curvearrowleft_\Box \mathcal{L} \stackrel{\text{def}}{=} \{\sigma\} \cdot (w' \curvearrowleft_\Box \mathcal{L})$ for each $\sigma \in \Sigma \setminus \{\Box\}$.

For two languages $\mathcal{L}, \mathcal{L}' \subseteq \Sigma^*$ and $\Box \in \Sigma_{int}$, the *M-substitution of $\Box$ by $\mathcal{L}'$ in $\mathcal{L}$*, written $\mathcal{L} \curvearrowleft_\Box \mathcal{L}'$, is defined as

$$\mathcal{L} \curvearrowleft_\Box \mathcal{L}' \stackrel{\text{def}}{=} \bigcup_{w \in \mathcal{L}} w \curvearrowleft_\Box \mathcal{L}'$$

Note that $\curvearrowleft_\Box$ is associative. Moreover, if $\{\Box\} \cap \mathcal{L} = \emptyset$, then $\{\Box\} \curvearrowleft_\Box \mathcal{L} = MWM(\mathcal{L})$.

▶ **Example 3.** Let $\Sigma_{call} = \{c_1, c_2\}$, $\Sigma_{ret} = \{r\}$, and $\Sigma_{int} = \{\Box\}$. Let us consider the languages $\mathcal{L} = \{c_1^n \,\Box\, \Box\, r^n \mid n \geq 1\}$ and $\mathcal{L}' = \{c_2\}^* \cdot \{r\}^*$. Then $\mathcal{L} \curvearrowleft_\Box \mathcal{L}'$ is given by $\{c_1^n\, c_2^m\, r^m\, c_2^k\, r^k\, r^n \mid n, m, k \geq 1\}$.

▶ **Definition 4** (M-closure and S-closure)**.** Given $\mathcal{L} \subseteq \Sigma^*$ and $\Box \in \Sigma_{int}$, the M-closure of $\mathcal{L}$ through $\Box$, denoted by $\mathcal{L}^{\curvearrowleft_\Box}$, is defined as:

$$\mathcal{L}^{\curvearrowleft_\Box} \stackrel{\text{def}}{=} \bigcup_{n \geq 0} \mathcal{L} \underbrace{\curvearrowleft_\Box (\mathcal{L} \cup \{\Box\}) \curvearrowleft_\Box \ldots \curvearrowleft_\Box (\mathcal{L} \cup \{\Box\})}_{n \text{ occurrences of } \curvearrowleft_\Box}.$$

The S-closure of $\mathcal{L}$ through $\Box$, denoted by $\mathcal{L}^{\circlearrowleft_\Box}$, is defined as $(MWM(\mathcal{L}))^{\curvearrowleft_\Box}$. Note that $\mathcal{L}^{\circlearrowleft_\Box}$ is contained in $MWM(\widetilde{\Sigma})$. The M-closure operator is a derived operator, since it can be expressed in terms of S-closure and M-substitution as follows:

$$\mathcal{L}^{\curvearrowleft_\Box} = \mathcal{L} \curvearrowleft_\Box (\mathcal{L}^{\circlearrowleft_\Box} \cup \{\Box\})$$

▶ **Example 5.** Let $\Sigma_{call} = \{c_1, c_2\}$, $\Sigma_{ret} = \{r_1, r_2\}$, and $\Sigma_{int} = \{\Box\}$. Let us consider the languages $\mathcal{L} = \{\Box, c_1 \,\Box\, r_1,\ c_2 \,\Box\, r_2\}$ and $\mathcal{L}' = \{c_1\, r_1,\ c_2\, r_2\}$. Then, $\mathcal{L}^{\curvearrowleft_\Box} \curvearrowleft_\Box \mathcal{L}' = \{c_{i_1}\, c_{i_2} \ldots c_{i_n}\, r_{i_n} \ldots r_{i_2}\, r_{i_1} \mid n \geq 1,\ i_1, \ldots, i_n \in \{1, 2\}\}$. One can easily show that there is no regular language $\mathcal{L}_{reg}$ such that $MWM(\mathcal{L}_{reg}) = \mathcal{L}^{\curvearrowleft_\Box} \curvearrowleft_\Box \mathcal{L}'$.

Now we show that VPL are closed under M-substitution, M-closure and S-closure.

▶ **Theorem 6.** *Let $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ and $\mathcal{P}' = \langle Q', q'_{in}, \Gamma', \Delta', F'\rangle$ be two NVPA over $\widetilde{\Sigma}$, and $\Box \in \Sigma_{int}$. Then, one can construct in polynomial time:*

1. *an NVPA accepting $(\mathcal{L}(\mathcal{P}))^{\circlearrowleft_\Box}$ with $|Q| + 2$ states and $|\Gamma| \cdot (|Q| + 2)$ stack symbols.*
2. *an NVPA accepting $\mathcal{L}(\mathcal{P}) \curvearrowleft_\Box \mathcal{L}(\mathcal{P}')$ with $|Q| + |Q'|$ states and $|\Gamma| + |\Gamma'| \cdot (|Q| + 1)$ stack symbols.*
3. *an NVPA accepting $(\mathcal{L}(\mathcal{P}))^{\curvearrowleft_\Box}$ with $2|Q| + 2$ states and $2|\Gamma| \cdot (|Q| + 1)$ stack symbols.*

Proof. Here, we sketch the construction of the NVPA for Condition 1. Fix an NVPA $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$. The construction consists of two steps. In the first step, we construct an NVPA $\mathcal{P}' = \langle Q', q'_{in}, \Gamma \cup \widehat{\Gamma}, \Delta', F'\rangle$ accepting $MWM(\mathcal{L}(\mathcal{P}))$, where $Q' \supseteq Q$, $|Q'| = |Q| + 2$, and $\widehat{\Gamma}$ is a fresh copy of Γ; moreover, each push transition from q'_{in} pushes onto the stack a symbol in $\widehat{\Gamma}$ and each internal transition is in Δ. In the second step, we construct an NVPA $\mathcal{P}''$ accepting $(\mathcal{L}(\mathcal{P}'))^{\circlearrowleft_\Box}$ with $|Q'|$ states and stack alphabet $\Gamma \cup \widehat{\Gamma} \cup Q \times \widehat{\Gamma}$. Hence, the result follows. Here, we informally describe the construction of $\mathcal{P}''$. Essentially, $\mathcal{P}''$ simulates $\mathcal{P}'$ step by step, but when $\mathcal{P}'$ performs an internal transition of the form $(q, \Box, q')$, then from the current state q, $\mathcal{P}''$ can choose to either process $\Box$ as $\mathcal{P}'$, or recursively process instead some guessed word $w \in \mathcal{L}(\mathcal{P}')^{\circlearrowleft_\Box}$ as follows. $\mathcal{P}''$ guesses a call c that is the initial symbol of w, and chooses a push transition $(q'_{in}, c, p, \gamma) \in \Delta'$ from the initial state of $\mathcal{P}'$ (the construction of $\mathcal{P}'$ ensures that $\gamma \in \widehat{\Gamma}$). In the same step, $\mathcal{P}''$ pushes onto the stack both γ and the target state $q' \in Q$ of the internal transition $(q, \Box, q')$ of $\mathcal{P}'$, and moves to state p. This compound step allows $\mathcal{P}''$ to guarantee that when the stack is popped on reading the matching return of c (corresponding to the last symbol of the guessed word w), $\mathcal{P}''$ can restart the simulation

of $\mathcal{P}'$ from the desired control state q'. Moreover, when the matching return of c is read, $\mathcal{P}''$ guarantee that the pair (q', γ) is popped from the stack if and only if from the current state of $\mathcal{P}''$, there is some pop transition of $\mathcal{P}'$ which pops γ and leads to some accepting state in F'. Note that since $\mathcal{P}'$ accepts only minimally well-matched words, the pair (q', γ) pushed onto the stack is eventually popped. ◀

3.1 VRE and Equivalence Between VRE and NVPA

▶ **Definition 7** (VRE). The syntax of VRE E over the pushdown alphabet $\widetilde{\Sigma}$ is defined as:

$$E := \emptyset \mid \varepsilon \mid \sigma \mid (E \cup E) \mid (E \cdot E) \mid E^* \mid (E \curvearrowleft_{\square} E) \mid E^{\circlearrowleft_{\square}} \mid E^{\curvearrowleft_{\square}}$$

where $\sigma \in \Sigma$ and $\square \in \Sigma_{int}$. A *pure* VRE is a VRE which does not contain occurrences of the M-closure operator $^{\curvearrowleft_{\square}}$. A VRE E denotes a language of finite words over Σ, written $\mathcal{L}(E)$, which is defined in the obvious way as follows:

- $\mathcal{L}(\emptyset) = \emptyset$, $\mathcal{L}(\varepsilon) = \{\varepsilon\}$, and $\mathcal{L}(\sigma) = \{\sigma\}$ for each $\sigma \in \Sigma$;
- $\mathcal{L}(E_1 \cup E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$, $\mathcal{L}(E_1 \cdot E_2) = \mathcal{L}(E_1) \cdot \mathcal{L}(E_2)$, and $\mathcal{L}(E^*) = [\mathcal{L}(E)]^*$;
- $\mathcal{L}(E_1 \curvearrowleft_{\square} E_2) = \mathcal{L}(E_1) \curvearrowleft_{\square} \mathcal{L}(E_2)$, $\mathcal{L}(E^{\circlearrowleft_{\square}}) = [\mathcal{L}(E)]^{\circlearrowleft_{\square}}$, and $\mathcal{L}(E^{\curvearrowleft_{\square}}) = [\mathcal{L}(E)]^{\curvearrowleft_{\square}}$.

As usual, the size $|E|$ of a VRE E is the length of the string describing E.

▶ Remark. By Definition 4, the M-closure operator is a derived operator. Hence, pure VRE and unrestricted VRE capture the same class of languages.

It is known [1] that for the class of regular languages (over a pushdown alphabet), NVPA can be exponentially more succinct than nondeterministic finite-state automata (NFA). We establish an analogous result for VRE and regular expressions.

▶ **Theorem 8.** *There are a pushdown alphabet $\widetilde{\Sigma}$ and a family $\{\mathcal{L}_n\}_{n\geq 1}$ of regular languages over $\widetilde{\Sigma}$ such that for each $n \geq 1$, $\mathcal{L}_n$ can be denoted by a VRE of size $O(n)$ and every regular expression denoting $\mathcal{L}_n$ has size at least $2^{\Omega(n)}$.*

Proof. Let $\widetilde{\Sigma} = \langle \Sigma_{call}, \Sigma_{ret}, \{\square\} \rangle$ with $\Sigma_{call} = \{c_1, c_2\}$ and $\Sigma_{ret} = \{r_1, r_2\}$. For $n \geq 1$, let $\mathcal{L}_n$ be the finite (hence, regular) language $\{c_{i_1} c_{i_2} \ldots c_{i_n} r_{i_n} \ldots r_{i_2} r_{i_1} \mid i_1, \ldots, i_n \in \{1, 2\}\}$. Evidently, $\mathcal{L}_n$ can be expressed by the VRE of size $O(n)$ given by $\underbrace{E \curvearrowleft_{\square} E \curvearrowleft_{\square} \ldots \curvearrowleft_{\square} E}_{n-1 \text{ times}} \curvearrowleft_{\square}$ $(c_1 \cdot r_1 \cup c_2 \cdot r_2)$, where $E = (c_1 \cdot \square \cdot r_1 \cup c_2 \cdot \square \cdot r_2)$. However, as shown in [1], any NFA accepting $\mathcal{L}_n$ requires at least 2^n states. Thus, since regular expressions can be converted in linear time into equivalent NFA, the result follows. ◀

In the following, we show that VRE and NVPA are effectively language equivalent. First, we recall the following known result [4].

▶ **Theorem 9** (From [4]). *Let $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F \rangle$ and $\mathcal{P}' = \langle Q', q'_{in}, \Gamma', \Delta', F' \rangle$ be two NVPA over $\widetilde{\Sigma}$. Then, one can construct in linear time:*

1. *an NVPA accepting $\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{P}')$ (resp. $\mathcal{L}(\mathcal{P}) \cdot \mathcal{L}(\mathcal{P}')$) with $|Q| + |Q'|$ states and $|\Gamma| + |\Gamma'|$ stack symbols.*
2. *an NVPA accepting $[\mathcal{L}(\mathcal{P})]^*$ with $2|Q|$ states and $2|\Gamma|$ stack symbols.*

By Theorems 6 and 9, a given VRE can be effectively and compositionally translated into an equivalent NVPA. However, due to Condition 3 in Theorem 6 (concerning the M-closure operator) and Condition 2 in Theorem 9 (concerning the Kleene closure operator), the translation can involve a singly exponential blow-up. In the next section, we show that this exponential blow-up is due essentially to the presence of the M-closure operator (in particular, we propose a quadratic time translation of *pure* VRE into equivalent NVPA).

▶ **Corollary 10.** *Given a VRE E, one can construct in* singly exponential time *an NVPA accepting $\mathcal{L}(E)$.*

Now, we show that any NVPA can be converted into an equivalent VRE. First, we need some additional notation. Let $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ be an NVPA over a pushdown alphabet $\widetilde{\Sigma}$. Given $p, p' \in Q$, a *summary* of $\mathcal{P}$ from p to p' is a run π of $\mathcal{P}$ over some word $w \in MWM(\widetilde{\Sigma})$ from a configuration of the form (p, β) to a configuration of the form (p', β') for some stack contents β and β'. Observe that if π is such a run over $w \in MWM(\widetilde{\Sigma})$ from (p, β) to (p', β'), then $\beta' = \beta$ and the portion of the stack corresponding to β is never read in π. In particular, there is also a run of $\mathcal{P}$ from $(p, \bot)$ to $(p', \bot)$ over w which uses the same transitions used by π. Given a run π of $\mathcal{P}$ over some word w and $\mathcal{S} \subseteq Q \times Q$, we say that the run π uses only *sub-summaries* from $\mathcal{S}$ whenever for all $q, q' \in Q$, if there is subrun of π which is a summary from q to q', then $(q, q') \in \mathcal{S}$. Given a finite alphabet Λ disjoint from Σ, we denote by $\widetilde{\Sigma}_\Lambda$ the pushdown alphabet $\langle \Sigma_{call}, \Sigma_{ret}, \Sigma_{int} \cup \Lambda\rangle$ obtained by interpreting the elements in Λ as internal actions.

▶ **Theorem 11.** *Given an NVPA $\mathcal{P}$, one can construct in single exponential time a VRE E such that $\mathcal{L}(E) = \mathcal{L}(\mathcal{P})$.*

Proof. Let $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$. We construct a finite alphabet Λ disjoint from Σ and a VRE E over $\widetilde{\Sigma}_\Lambda$ denoting $\mathcal{L}(\mathcal{P})$; the additional symbols in Λ are used only as parameters for intermediate substitutions. The alphabet Λ is given by $\{\Box_{pp'} \mid p, p' \in Q\}$. Moreover, we define $\mathcal{P}_\Lambda = \langle Q, q_{in}, \Gamma, \Delta_\Lambda, F\rangle$ as the NVPA over $\widetilde{\Sigma}_\Lambda$ obtained from $\mathcal{P}$ by adding for each $(p, p') \in Q \times Q$, the internal transition $(p, \Box_{pp'}, p')$. Given $q, q' \in Q$, $\mathcal{S} \subseteq Q \times Q$, and $\Lambda' \subseteq \Lambda$, we define $R(q, q', \mathcal{S}, \Lambda')$ as the language of finite words w over $\Sigma_{\Lambda'}$ ($\Sigma_{\Lambda'}$ is the support of $\widetilde{\Sigma}_{\Lambda'}$) such that there is a run of $\mathcal{P}_\Lambda$ over w from $(q, \bot)$ to some configuration of the form (q', β) which uses only sub-summaries from $\mathcal{S}$.[2] By construction, $\mathcal{L}(\mathcal{P})$ is the union of the sets $R(q, q', Q \times Q, \emptyset)$ such that $q = q_{in}$ and $q' \in F$. Thus, the theorem follows from the fact that for all $q, q' \in Q$, $\mathcal{S} \subseteq Q \times Q$, and $\Lambda' \subseteq \Lambda$, the languages $R(q, q', \mathcal{S}, \Lambda')$ and $WM(R(q, q', \mathcal{S}, \Lambda'))$ can be effectively denoted by VRE of sizes singly exponential in the size of $\mathcal{P}$, where $WM(R(q, q', \mathcal{S}, \Lambda')) \stackrel{\text{def}}{=} R(q, q', \mathcal{S}, \Lambda') \cap WM(\widetilde{\Sigma})$. The proof of this fact proceeds by induction on the cardinality of the finite set $\mathcal{S}$.

Base case: $\mathcal{S} = \emptyset$. The proof of the base case is simple. In particular, the languages $R(q, q', \emptyset, \Lambda')$ and $WM(R(q, q', \emptyset, \Lambda'))$ are regular.

Induction step: $\mathcal{S} = \mathcal{S}' \cup \{(p, p')\}$ with $(p, p') \notin \mathcal{S}'$. Let $P_{p \to p'}$ be the set:

$$\{(s, c, r, s') \in Q \times \Sigma_{call} \times \Sigma_{ret} \times Q \mid \text{ there is } \gamma \in \Gamma . (p, c, s, \gamma), (s', r, \gamma, p') \in \Delta\}$$

So, $P_{p \to p'}$ is the set of tuples (s, c, r, s') such that there is a push transition from p to s reading the call c and a matching pop transition from s' to p' reading r. Moreover, let $S(p, p', \mathcal{S}' \cup \{(p, p')\}, \Lambda')$ be the language over $\Sigma_{\Lambda'}$ defined as follows:

$$S(p, p', \mathcal{S}' \cup \{(p, p')\}, \Lambda') := \Big(\Big[\bigcup_{(s,c,r,s') \in P_{p \to p'}} \{c\} \cdot WM(R(s, s', \mathcal{S}', \Lambda' \cup \{\Box_{pp'}\})) \cdot \{r\} \Big]^{\curvearrowright \Box_{pp'}} \Big) \curvearrowleft \Box_{pp'} \Big[\bigcup_{(s,c,r,s') \in P_{p \to p'}} \{c\} \cdot WM(R(s, s', \mathcal{S}', \Lambda')) \cdot \{r\} \Big]$$

[2] note that if $w \in WM(\widetilde{\Sigma}_{\Lambda'})$, then $\beta = \bot$.

Note that $S(p,p',\mathcal{S}' \cup \{(p,p')\},\Lambda')$ represents the set of words $w \in MWM(\widetilde{\Sigma}_{\Lambda'})$ such that there is a summary of $\mathcal{P}_\Lambda$ over w from p to p' which uses only sub-summaries from $\mathcal{S}' \cup \{(p,p')\}$. By the induction hypothesis, the sets $WM(R(s,s',\mathcal{S}',\Lambda' \cup \{\square_{pp'}\}))$ and $WM(R(s,s',\mathcal{S}',\Lambda'))$ used in the definition of $S(p,p',\mathcal{S}' \cup \{(p,p')\},\Lambda')$ can be effectively denoted by VRE. Thus, one can construct a VRE over $\widetilde{\Sigma}_\Lambda$ denoting the language $S(p,p',\mathcal{S}' \cup \{(p,p')\},\Lambda')$. Now, we observe that:

$$\begin{aligned} WM(R(q,q',\mathcal{S}' \cup \{(p,p')\},\Lambda')) = {} & WM(R(q,q',\mathcal{S}',\Lambda')) \cup \\ & WM(R(q,q',\mathcal{S}',\Lambda' \cup \{\square_{pp'}\})) \curvearrowleft_{\square_{pp'}} S(p,p',\mathcal{S}' \cup \{(p,p')\},\Lambda') \\ R(q,q',\mathcal{S}' \cup \{(p,p')\},\Lambda') = {} & R(q,q',\mathcal{S}',\Lambda') \cup \\ & R(q,q',\mathcal{S}',\Lambda' \cup \{\square_{pp'}\}) \curvearrowleft_{\square_{pp'}} S(p,p',\mathcal{S}' \cup \{(p,p')\},\Lambda') \end{aligned}$$

Thus, by the induction hypothesis, it holds that the languages $WM(R(q,q',\mathcal{S}' \cup \{(p,p')\},\Lambda'))$ and $R(q,q',\mathcal{S}' \cup \{(p,p')\},\Lambda')$ can be effectively denoted by VRE. Moreover, by expanding recursively the above equalities until the base case (the number of iterations is at most $|Q|^2$), we deduce that each of the constructed VRE has size singly exponential in the size of the NVPA $\mathcal{P}$. This concludes the proof of the theorem. ◀

By Corollary 10 and Theorem 11, we obtain the following result.

▶ **Corollary 12.** *(Pure) Visibly Rational Expressions capture the class of VPL.*

4 Pure VRE

In this section, first, we show that pure VRE can be compositionally translated in *quadratic time* into equivalent NVPA. The key of the proposed efficient and elegant translation is represented by a subclass of NVPA, we call *strong* NVPA. Then, in Subsection 4.1, we establish the exact complexity of some language decision problems for pure VRE. Fix a pushdown alphabet $\widetilde{\Sigma}$. In the following, we use an additional special stack symbol $\widehat{\perp}$.

▶ **Definition 13.** *A* strong *NVPA over $\widetilde{\Sigma}$ is an NVPA $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ over $\widetilde{\Sigma}$ such that $\widehat{\perp} \in \Gamma$ and the following holds:*
Initial State Requirement*: $q_{in} \notin F$ and there are no transitions leading to q_{in}.*
Final State Requirement*: there are no transitions from accepting states.*
Push Requirement*: every push transition from the initial state q_{in} pushes onto the stack the special symbol $\widehat{\perp}$.*
Pop Requirement*: for all $q,p \in Q$ and $r \in \Sigma_{ret}$, $(q,r,\perp,p) \in \Delta$ iff $(q,r,\widehat{\perp},p) \in \Delta$.*
Well-formed (semantic) Requirement: *for all $w \in \mathcal{L}(\mathcal{P})$, every initialized accepting run of $\mathcal{P}$ over w leads to a configuration whose stack content is in $\{\widehat{\perp}\}^* \cdot \perp$.*
Note that the initial state requirement implies that $\varepsilon \notin \mathcal{L}(\mathcal{P})$.

The push requirement is used in particular to implement in an efficient way M-substitution and S-closure. Moreover, the pop requirement ensures that pop operations which pop the special stack symbol $\widehat{\perp}$ have the same effect as popping the empty stack (i.e., the stack containing just the special bottom symbol $\perp$). This requirement and the well-formed requirement are used in particular to implement in an efficient way concatenation and Kleene closure. In the following, we first show that strong NVPA are "efficiently" closed under union, concatenation, and Kleene closure.

▶ **Theorem 14.** *Let $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ and $\mathcal{P}' = \langle Q', q'_{in}, \Gamma', \Delta', F'\rangle$ be two strong NVPA over $\widetilde{\Sigma}$. Then, one can construct in linear time*

1. *a strong NVPA accepting* $\mathcal{L}(\mathcal{P}) \cup \mathcal{L}(\mathcal{P}')$ *(resp.,* $\mathcal{L}(\mathcal{P}) \cdot \mathcal{L}(\mathcal{P}')$*) with* $|Q| + |Q'| + 1$ *states and* $|\Gamma| + |\Gamma'| - 1$ *stack symbols, and*
2. *a strong NVPA accepting* $[\mathcal{L}(\mathcal{P})]^* \setminus \{\varepsilon\}$ *with* $|Q| + 1$ *states and* $|\Gamma|$ *stack symbols.*

Proof. We prove Condition 2 (Condition 1 is simpler). Let $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ be a strong NVPA over $\widetilde{\Sigma}$ an q'_{in} be a fresh control state. Define $\mathcal{P}' = \langle Q \cup \{q'_{in}\}, q'_{in}, \Gamma, \Delta', F\rangle$, where Δ' is obtained from Δ by adding new transitions as follows. First, for each transition $t \in \Delta$ leading to an accepting state, we add the transition obtained from t by replacing the target state of t with the initial state q_{in}. Let Δ_0 be the resulting set of transitions. Then, Δ' is obtained from Δ_0 by adding the following transitions: for each transition $t \in \Delta_0$ from the initial state q_{in}, we add the transition obtained from t by replacing the source state of t with the new initial state q'_{in}. Note that the construction is identical to the classical one used for regular languages. Now, we prove that $\mathcal{P}'$ is a *strong* NVPA accepting $[\mathcal{L}(\mathcal{P})]^* \setminus \{\varepsilon\}$. Hence, the result follows. Since $\mathcal{P}$ is a strong NVPA, by construction, it follows that $\mathcal{P}'$ satisfies the initial and final state requirements and the push and pop requirements of Definition 13. Thus, it remains to show that $\mathcal{L}(\mathcal{P}') = [\mathcal{L}(\mathcal{P})]^* \setminus \{\varepsilon\}$ and $\mathcal{P}'$ satisfies the well-formed requirement.

Here, we show that $\mathcal{L}(\mathcal{P}') \subseteq [\mathcal{L}(\mathcal{P})]^* \setminus \{\varepsilon\}$ and $\mathcal{P}'$ satisfies the well formed requirement. Let $w \in \mathcal{L}(\mathcal{P}')$ (note that $w \neq \varepsilon$ since $\mathcal{P}'$ satisfies the initial state requirement) and π be an initialized accepting run of $\mathcal{P}'$ over w of the form $(q'_{in}, \bot) \xrightarrow{w} (q_{acc}, \beta)$ for some stack content β and $q_{acc} \in F$. We need to show that $w \in [\mathcal{L}(\mathcal{P})]^* \setminus \{\varepsilon\}$ and $\beta \in \{\widehat{\bot}\}^* \cdot \bot$. By construction, w is of the form $w = w_1 \cdot \ldots \cdot w_n$ for some $n \geq 1$, such that $w_1, \ldots, w_n$ are non-empty and there are runs $\pi_1, \ldots, \pi_n$ of $\mathcal{P}$ over $w_1, \ldots, w_n$, respectively, of the form

$$\pi_1 = (q_{in}, \bot) \xrightarrow{w_1} (p_1, \beta_1), \quad \pi_2 = (q_{in}, \beta_1) \xrightarrow{w_2} (p_2, \beta_2), \quad \ldots \ ,$$
$$\pi_n = (q_{in}, \beta_{n-1}) \xrightarrow{w_n} (p_n, \beta_n)$$

where $p_i \in F$ for each $1 \leq i \leq n$, and $\beta = \beta_n$. We show by induction on i that $w_i \in \mathcal{L}(\mathcal{P})$ and $\beta_i \in \{\widehat{\bot}\}^* \cdot \bot$ for all $1 \leq i \leq n$, hence the result follows. Since π_1 is an initialized accepting run of $\mathcal{P}$ over w_1 and $\mathcal{P}$ satisfies the well-formed requirement, the result for the base case holds. For the induction step, let us consider the run $\pi_i = (q_{in}, \beta_{i-1}) \xrightarrow{w_i} (p_i, \beta_i)$ with $i > 1$. By the induction hypothesis, $\beta_{i-1} \in \{\widehat{\bot}\}^* \cdot \bot$. Moreover, β_i is of the form $\beta_i = \beta'_i \cdot \{\widehat{\bot}\}^m \cdot \bot$ for some $m \geq 0$, where β'_i consists of the symbols pushed on the stack along π_i on reading the unmatched call positions of w_i. Since $\mathcal{P}$ satisfies the pop requirement, we easily deduce that there is also an *initialized accepting* run of $\mathcal{P}$ over w_i of the form $\pi_i = (q_{in}, \bot) \xrightarrow{w_i} (p_i, \beta'_i \cdot \bot)$. Since $\mathcal{P}$ satisfies the well-formed requirement, $\beta'_i \in \{\widehat{\bot}\}^*$. Hence, $\beta_i \in \{\widehat{\bot}\}^* \cdot \bot$, and we are done. This concludes the proof of the theorem. ◀

Next we show that strong NVPA are "efficiently" closed under M-substitution and S-closure. For this, we need the following preliminary result.

▶ **Lemma 15.** *Let* $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ *be a strong NVPA over* $\widetilde{\Sigma}$*. Then, one can construct in linear time a strong NVPA over* $\widetilde{\Sigma}$ *accepting* $MWM(\mathcal{L}(\mathcal{P}))$ *with* $|Q|$ *states and* $|\Gamma| + 1$ *stack symbols.*

▶ **Theorem 16.** *Let* $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ *and* $\mathcal{P}' = \langle Q', q'_{in}, \Gamma', \Delta', F'\rangle$ *be two strong NVPA over* $\widetilde{\Sigma}$*, and* $\Box \in \Sigma_{int}$*. Then, one can construct in linear time: (1) a strong NVPA accepting* $(\mathcal{L}(\mathcal{P}))^{\circlearrowleft_\Box}$ *with* $|Q|$ *states and* $|Q| + |\Gamma| + 1$ *stack symbols, and (2) a strong NVPA accepting* $\mathcal{L}(\mathcal{P}) \curvearrowleft_\Box \mathcal{L}(\mathcal{P}')$ *with* $|Q| + |Q'|$ *states and* $|\Gamma| + |\Gamma'| + |Q|$ *stack symbols.*

Proof. Here, we focus on Condition (1). Given a strong NVPA $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ over $\widetilde{\Sigma}$ such that $\mathcal{L}(\mathcal{P}) \subseteq MWM(\widetilde{\Sigma})$, we construct a strong NVPA $\mathcal{P}'$ accepting $(\mathcal{L}(\mathcal{P}))^{\circlearrowleft_\Box}$ with

$|Q|$ states and $|Q| + |\Gamma|$ stack symbols. Hence, by Lemma 15, Condition (1) in the theorem follows. W.l.o.g. we assume that Q and Γ are disjoint and all the transitions from the initial state are push transitions. The NVPA $\mathcal{P}'$ is given by $\mathcal{P}' = \langle Q, q_{in}, \Gamma \cup Q, \Delta', F\rangle$, where Δ' is obtained from Δ by adding the following transitions:

- *New Push transitions*: for each internal transition $(q, \Box, p) \in \Delta$ – note that $q \neq q_{in}$ – and for each push transition from the initial state of the form $(q_{in}, c, q', \widehat{\perp}) \in \Delta$, we add the new push transition (q, c, q', p).
- *New Pop transitions*: for each pop transition $(q, r, \widehat{\perp}, q_{acc}) \in \Delta$ which pops the special stack symbol $\widehat{\perp}$ and leads to an accepting state $q_{acc} \in F$, we add for each $p \in Q \setminus \{q_{in}\}$, the new pop transition (q, r, p, p).

Correctness of the construction directly follows from the following claim.

Claim: $\mathcal{P}'$ is a strong NVPA over $\widetilde{\Sigma}$ accepting $[\mathcal{L}(\mathcal{P})]^{\circlearrowleft\Box}$. ◀

Now, we can prove the main result of this section.

▶ **Theorem 17.** *Let E be a* pure *VRE. Then, one can construct in quadratic time an NVPA $\mathcal{P}$ accepting $\mathcal{L}(E)$ with at most $|E| + 1$ states and $|E|^2$ stack symbols.*

Proof. Since one can trivially check in linear time whether $\varepsilon \in \mathcal{L}(E)$, it suffices to show that one can construct in quadratic time a *strong* NVPA accepting $\mathcal{L}(E) \setminus \{\varepsilon\}$ with at most $|E| + 1$ states and $|E|^2$ stack symbols. The proof is by induction on $|E|$. The base case is trivial. For the induction step, the result easily follows from the induction hypothesis and Theorems 14 and 16. As example, we illustrate the case where $E = E_1 \curvearrowleft_{\Box} E_2$. By the induction hypothesis, one can construct two strong NVPA $\mathcal{P}_1 = \langle Q_1, q^1_{in}, \Gamma_1, \Delta_1, F_1\rangle$ and $\mathcal{P}_2 = \langle Q_2, q^2_{in}, \Gamma_2, \Delta_2, F_2\rangle$ accepting $\mathcal{L}(E_1) \setminus \{\varepsilon\}$ and $\mathcal{L}(E_2) \setminus \{\varepsilon\}$, respectively. Moreover, $|Q_1| \leq |E_1|+1$, $|Q_2| \leq |E_2|+1$, $|\Gamma_1| \leq |E_1|^2$, and $|\Gamma_2| \leq |E_2|^2$. By Theorem 16, one can construct in linear time a strong NVPA $\mathcal{P} = \langle Q, q_{in}, \Gamma, \Delta, F\rangle$ accepting $(\mathcal{L}(E_1)\setminus\{\varepsilon\})\curvearrowleft_{\Box}(\mathcal{L}(E_2)\setminus\{\varepsilon\}) = \mathcal{L}(E) \setminus \{\varepsilon\}$. Moreover, $|Q| = |Q_1| + |Q_2|$ and $|\Gamma| = |\Gamma_1| + |\Gamma_2| + |Q_1|$. Hence, $|\Gamma| \leq |E_1|^2 + |E_2|^2 + |E_1| + 1 \leq (|E_1| + |E_2| + 1)^2 = |E|^2$ and $|Q| \leq |E_1| + |E_2| + 2 = |E| + 1$, and the result follows. ◀

4.1 Decision Problems for pure VRE

In this section, we show the following result.

▶ **Theorem 18.** *The universality, inclusion, and language equivalence problems for pure VRE are EXPTIME-complete.*

Sketched proof. The upper bounds directly follow from Theorem 17 and EXPTIME-completeness of universality, inclusion, and equivalence for NVPA [4]. For the lower bounds, it is sufficient to show EXPTIME-hardness for the universality problem. This is proved by a polynomial time reduction from the word problem for polynomial space bounded alternating Turing Machines (TM) with a binary branching degree, which is a well-known EXPTIME-complete problem [11]. Fix such a machine $\mathcal{M}$ with input alphabet A and set of states Q. Since $\mathcal{M}$ is polynomial space bounded, there is an integer constant $k \geq 1$ such that for each $\alpha \in A^*$, the space needed by $\mathcal{M}$ on the input α is bounded by $|\alpha|^k$. Fix an input α and let $n = |\alpha|$. W.l.o.g. we can assume that $k = 1$, $n > 1$, and each (reachable) TM configuration (from the fixed input α) can be described by a word in $A^* \cdot (Q \times A) \cdot A^*$ of length exactly n. Let T_{full} be the configuration-labeled binary tree corresponding to the unwinding of $\mathcal{M}$ from the initial configuration associated with the input α. A computation tree T of $\mathcal{M}$

(over α) is a *finite* tree obtained from T_{full} by pruning subtrees rooted at children of nodes labeled by existential configurations; T is accepting if each leaf is labeled by an accepting TM configuration. $\mathcal{M}$ accepts α if there is an accepting computation tree (over α). We use a standard encoding of computation trees T by minimally well-matched words w_T over a suitable pushdown alphabet $\widetilde{\Sigma}$ [9, 4], where the given tree T is processed in depth-first order. This encoding ensures the following crucial property: for all nodes x and y of T labeled by TM configurations C_x and C_y such that y is the child of x, there is a subword of w_T encoding $C_x \cdot C_y$ or its reverse. Let $Codes(\alpha)$ be the set of words $w \in MWM(\widetilde{\Sigma})$ encoding accepting computation trees (over α). Then, we show that it is possible to construct in time polynomial in n and the size of $\mathcal{M}$ a pure VRE over $\widetilde{\Sigma}$ which denotes the language $\Sigma^* \setminus Codes(\alpha)$. Hence, the result follows. ◀

5 ω-Visibly Rational Expressions (ω-VRE)

In this section, we introduce the class of ω-Visibly Rational Expressions (ω-VRE) and provide a Büchi-like theorem for ω-VPL in terms of ω-VRE. Fix a pushdown alphabet $\widetilde{\Sigma}$. For a language $\mathcal{L}$ of finite words over Σ, we denote by $\mathcal{L}^\omega$ the standard ω-Kleene closure of $\mathcal{L}$.

▶ **Definition 19.** The syntax of ω-VRE I over $\widetilde{\Sigma}$ is inductively defined as follows:

$$I := (E)^\omega \mid (I \cup I) \mid (E \cdot I)$$

where E is a VRE over $\widetilde{\Sigma}$. Note that ω-VRE are defined similarly to ω-regular expressions. An ω-VRE I is *pure* if every VRE subexpression is pure. An ω-VRE I denotes a language of infinite words over Σ, written $\mathcal{L}(I)$, defined in the obvious way: $\mathcal{L}(E^\omega) = [\mathcal{L}(E)]^\omega$, $\mathcal{L}(I \cup I') = \mathcal{L}(I) \cup \mathcal{L}(I')$, and $\mathcal{L}(E \cdot I) = \mathcal{L}(E) \cdot \mathcal{L}(I)$.

We show that ω-VRE capture the class of ω-VPL. For this, we need the following preliminary result establishing that ω-VPL can be expressed in terms of VPL in the same way as ω-regular languages can be expressed in terms of regular languages.

▶ **Theorem 20.** *Let $\mathcal{L}$ be a ω-VPL with respect to $\widetilde{\Sigma}$. Then, there are $n \geq 1$ and VPL $\mathcal{L}_1, \mathcal{L}'_1, \ldots, \mathcal{L}_n, \mathcal{L}'_n$ with respect to $\widetilde{\Sigma}$ such that $\mathcal{L} = \bigcup_{i=1}^{i=n} \mathcal{L}_i \cdot (\mathcal{L}'_i)^\omega$. Moreover, the characterization is constructive.*

Since ω-VPL are effectively closed under ω-Kleene closure and under (left) concatenation with VPL (and the constructions can be done in linear time) [4], by Corollary 10, it follows that ω-VRE can be converted into equivalent ω-NVPA in single exponential time. Moreover, by using *strong* NVPA and constructions very similar to those used in the proof of Theorem 14, we can show that *pure* ω-VRE can be converted into equivalent Büchi ω-NVPA in quadratic time. Thus, by Theorem 20 we obtain the following result.

▶ **Theorem 21.** *(Pure) ω-VRE capture the class of ω-VPL. Moreover, pure ω-VRE can be converted in quadratic time into equivalent Büchi ω-NVPA.*

6 Conclusion

In this paper we have provided a Kleene/Büchi theorem for VPL. From a theoretical point of view, there are some interesting open questions. For example, the succinctness gap between VRE and NVPA (it is well-known that NFA are exponentially more succinct than regular expressions). From a practical point of view, it remains to be seen whether VRE are useful

as a specification language for nested word search and for XML schemas. Another line of future work is the combination of VRE with temporal logics for nested words (like CaRet [2]), as done for word regular languages [13, 17].

References

1 R. Alur. Marrying words and trees. In *Proc. 26th PODS*, pages 233–242. ACM, 2007.

2 R. Alur, K. Etessami, and P. Madhusudan. A Temporal Logic of Nested Calls and Returns. In *Proc. 10th TACAS*, volume 2988 of LNCS, pages 467–481. Springer, 2004.

3 R. Alur, V. Kumar, P. Madhusudan, and M. Viswanathan. Congruences for Visibly Pushdown Languages. In *Proc. 32nd ICALP*, volume 3580 of LNCS, pages 1102–1114. Springer, 2005.

4 R. Alur and P. Madhusudan. Visibly Pushdown Languages. In *Proc. 36th STOC*, pages 202–211. ACM, 2004.

5 R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3), 2009.

6 M. Arenas, P. Barceló, and L. Libkin. Regular Languages of Nested Words: Fixed Points, Automata, and Synchronization. In *Proc. 34th ICALP*, volume 4596 of LNCS, pages 888–900. Springer, 2007.

7 T. Ball and S.K. Rajamani. Bebop: a symbolic model checker for boolean programs. In *Proc. 7th SPIN*, volume 1885 of LNCS, pages 113–130. Springer, 2000.

8 C. Bolduc and B. Ktari. Visibly Pushdown Kleene Algebra and Its Use in Interprocedural Analysis of (Mutually) Recursive Programs. In *Proc. 11th RelMiCS*, volume 5827 of LNCS, pages 44-58. Springer, 2009.

9 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model-checking. In *Proc. 8th CONCUR*, volume 1243 of LNCS, pages 135–150. Springer, 1997.

10 L. Bozzelli. Alternating automata and a temporal fixpoint calculus for visibly pushdown languages. In *Proc. 18th CONCUR*, volume 4703 of LNCS, pages 476–491. Springer, 2007.

11 A.K. Chandra, D. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

12 A. Cyriac. Temporal Logics for Concurrent Recursive Programs. Rapport de Master, Master Parisien de Recherche en Informatique, Paris, France, September 2010.

13 D. Fisman, C. Eisner, and J. Havlicek. Formal syntax and Semantics of PSL: Appendix B of Accellera Property Language Reference Manual, Version 1.1. March, 2004.

14 J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages and computation.* Addison-Wesley, 1979.

15 S.C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In *Automata Studies*, volume 34, pages 3–41. Princeton University Press, 1956.

16 V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *Proc. 16th WWW*, pages 1053–1062. ACM, 2007.

17 M. Leucker and C. Sánchez. Regular Linear Temporal Logic. In *Proc. 4th ICTAC*, volume 4711 of LNCS, pages 291–305. Springer, 2007.

18 C. Löding, P. Madhusudan, and O. Serre. Visibly Pushdown Games. In *Proc. 24th FSTTCS*, volume 3328 of LNCS, pages 408–420. Springer, 2004.

19 P. Madhusudan and M. Viswanathan. Query Automata for XML Nested Words. In *Proc. 34th MFCS*, volume 5734 of LNCS, pages 561–573. Springer, 2009.

20 C. Pitcher. Visibly Pushdown Expression Effects for XML Stream Processing. In *Proc. PLAN-X*, pages 1–14. ACM, 2005.

21 A. Thomo, S. Venkatesh, and Y.Y. Ye. Visibly Pushdown Transducers for Approximate Validation of Streaming XML. In *Proc. 5th FoIKS*, volume 4932 of LNCS, pages 219-238. Springer, 2008.

22 K. Thompson. Regular expression search algorithm. *Comm. of the ACM*, 11(6):419–422, 1968.

Safety Verification of Communicating One-Counter Machines

Alexander Heußner[1], Tristan Le Gall[2], and Grégoire Sutre[3]

1 **ULB, Brussels, Belgium & University of Bamberg, Bamberg, Germany**
2 **CEA, LIST, DILS/LMeASI, Gif-sur-Yvette, France**
3 **Univ. Bordeaux & CNRS, LaBRI, UMR 5800, Talence, France**

Abstract

In order to verify protocols that tag messages with integer values, we investigate the decidability of the reachability problem for systems of communicating one-counter machines. These systems consist of local one-counter machines that asynchronously communicate by exchanging the value of their counters via, a priori unbounded, Fifo channels. This model extends communicating finite-state machines (Cfsm) by infinite-state local processes and an infinite message alphabet. The main result of the paper is a complete characterization of the communication topologies that have a solvable reachability question. As already Cfsm exclude the possibility of automatic verification in presence of mutual communication, we also consider an under-approximative approach to the reachability problem, based on rendezvous synchronization.

1998 ACM Subject Classification F.1.1 Models of Computation, D.2.4 Program Verification

Keywords and phrases Counter Machines, Fifo Channels, Reachability Problem, Data Words

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.224

1 Introduction

One of the most challenging and imperative problems in computer science today is the verification of the nowadays ubiquitous distributed systems, as these are increasingly applied in vital and sensitive areas. Such systems consist of several processes that asynchronously exchange data over a network topology. A well-established model, known as *communicating finite-state machines* (Cfsm), combines local finite-state machines with point-to-point, unbounded Fifo queues that pass messages from a finite alphabet. Cfsm laid the foundation for a family of infinite-state models parametrized by the computational power of the local machines, such as communicating Petri nets [10] and pushdown systems [14, 13].

However, basic safety verification questions, like reachability, are known to be undecidable for Cfsm already on simple topologies [6, 17]. One important line of current research is the influence of the underlying communication topology to these verification questions when we restrict the interplay between communication and the local machine's power [14, 7, 13]. In this paper, we extend this research towards the verification of communicating machines that locally use counters and can exchange these via message passing, thus introducing two additional sources of infinity to Cfsm's unbounded channels. Infinite message alphabets are demanded in practice to model protocols based on (*a priori* unbounded) sequence numbers.

Motivating Example. A simple sliding window protocol is depicted in Figure 1. A *sender* transmits a sequence number (ignoring additional data) to a *receiver* that advances the expected sequence number if it got the right message, demands to resend the expected

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 224–235

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

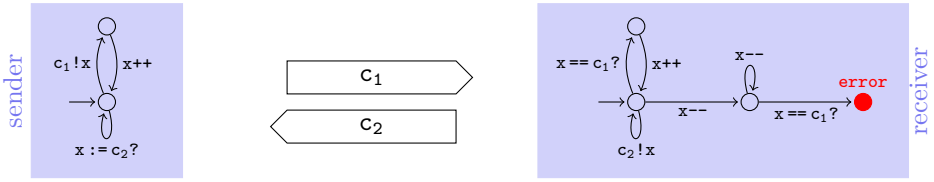


Figure 1 A simple sliding window protocol: sender on the left, receiver on the right.

message, or fails if the sequence number was already received. Checking the correctness of such protocols (here, whether the `error` state is reachable) is the main topic of this paper.

Contributions. We present the formal model of *systems of communicating one-counter machines*. This model is parametrized by a communication topology, specifying point-to-point Fifo channels between processes. Processes are one-counter machines that can send or receive the contents of their local counter. We consider an extension of one-counter machines where tests are not limited to zero-tests $x = 0$, but can be any unary Presburger predicate $\varphi(x)$. Channels are a priori unbounded, and messages are natural numbers. Different ways of relating these messages to the machine's local counters are investigated. As our main result, we establish a complete classification of the topologies over which the reachability problem for systems of communicating one-counter machines is decidable. The underlying proof relies, on the one hand, on a reduction from the well-known undecidability of the reachability problem for two-counters Minsky machines. On the other hand, we use a reduction technique that inductively combines one-counter machines along a hierarchical order, which is based on the topology. This way, the reachability problem is reduced to the case of two processes that are connected by one channel. We show that the reachability problem is decidable in this setting.

Our decidability results are based on summarizing the behavior of a process between each communication action. Recall that the reachability relation of a one-counter machine is definable in Presburger arithmetic (see, e.g., [11]). But Presburger-definable binary relations are not closed under transitive closure, which makes them unsuitable for our summarization-based approach. As key ingredient to our proofs, we exhibit a class of binary Presburger predicates that corresponds exactly to one-counter reachability relations. Our characterization entails that this class is effectively closed under transitive closure, and that one-counter reachability relations are effectively closed under intersection.

As the undecidable topologies include cyclic architectures, that nevertheless are important in practice to permit mutual communication, we also consider an under-approximative approach based on *eager* runs, i.e., runs where a send action is directly followed by its reception. We characterize the strongly-connected topologies that have a decidable eager-reachability problem. In particular, the topology of our motivating example, which is a cycle of length two, allows to decide the verification problem (for eager runs).

Related Works. The basic undecidability result for Cfsm [6] is the corner stone for most ongoing research on models based local machines that communicate over Fifo channels. Prominent approaches to regain decidability for reachability/safety are restrictions on the size of the channels or the message alphabets (already in [6, 17]), as well as the focus on lossy channel systems [9, 1]. Recent research dealt with the influence of the underlying topology on decidability questions, e.g., systems mixing lossy and perfect channels [7]. Communicating pushdown machines focus on a typing of channel ends that forces the decoupling of pushdown and channel actions [14, 13]. Restricting the local pushdown alphabet to a singleton, but

extending the finite message alphabet to an infinite one leads in our case to an incomparable model. However, we similarly arrive at favorable decidability results for tree-like architectures, which are more restricted than those in [13] even when regarding only eager communication.

CFSM-style systems with *infinite* message alphabets were discussed in [15], but this work focused on the definition of a static analysis technique, and thus the practical implementation of verification algorithms. Also closely related are data words and their different underlying automata models that rely on an infinite input/output alphabet and local registers [3, 4]. However, these automata only allow to use an equality test on the infinite data alphabet and not to modify and test registers like counters do.

Counter machines are a classical formalism in computer science [16]. Besides the *two-counters* (Minsky) machines, which are Turing-complete, the verification of *one-counter* automata has gained a renewed interest recently [8, 12, 2]. Using one-counter automata with Presburger tests also appears in [5], yet only as symbolic representation of reachability sets and not as operational model for the underlying programs.

Outline. We introduce systems of communicating one-counter machines in Section 2. Section 3 presents our main result: the characterization of communication topologies that have a solvable reachability question. The proof of the positive case is provided in Section 4. Section 5 presents preliminary results on the decidability of the reachability question when we only consider eager runs. Some conclusions and perspectives are given in Section 6.

2 Systems of Communicating One-Counter Machines

Given a (possibly infinite) alphabet M, let M^* denote the set of all finite *words* over M, $\varepsilon \in M^*$ the *empty* word, and $u \cdot v$ the *concatenation* of two words $u, v \in M^*$. For a set of values X and a finite set of indices I, we write X^I for the set of all mappings from I to X. Such mappings may be interpreted as I-indexed X-valued *vectors*. Let x^i denote the i-th component of a vector $\boldsymbol{x} \in X^I$. Two constant vectors are introduced, for convenience: $\mathbf{0} \in \mathbb{N}^I$, which maps every index to 0, and $\boldsymbol{\varepsilon} \in (M^*)^I$, which maps every index to ε.

Communication Topologies. In our framework, channels are point-to-point. Each channel c has a source endpoint $\mathsf{src}(c)$, and a destination endpoint $\mathsf{dst}(c)$. These endpoints are pairs $(p, *)$ where p is the process communicating at the endpoint, and $* \in \{\bullet, \circ\}$ is the communication type of the endpoint. We introduce the types $\bullet$ and $\circ$ to model two communication policies that relate the message and the local counter of a machine before and after communication on an endpoint. We assert that $\circ$ is more restrictive than $\bullet$, namely, that the value of the local counter is "lost" by a communication with type $\circ$. This difference is formalized in the semantics introduced subsequently. First, let us formally define communication topologies.

▶ **Definition 2.1.** A *topology* is a quadruple $\mathcal{T} = \langle P, C, \mathsf{src}, \mathsf{dst} \rangle$ where P is a finite, non-empty set of *processes*, C is a finite, possibly empty set of *channels*, $\mathsf{src} : C \to P \times \{\bullet, \circ\}$ is a *source* mapping, and $\mathsf{dst} : C \to P \times \{\bullet, \circ\}$ is a *destination* mapping.

For better readability, we slightly abuse notation by identifying an endpoint $(p, *)$ with its process p or its type $*$, depending on the context. For instance, we write $\mathsf{src}(c) = p$ instead of $\mathsf{src}(c) = (p, *)$ for some $* \in \{\bullet, \circ\}$. Given a process $p \in P$, we let $C(p)$ denote the set of all channels with source or destination p. Formally, $C(p) = \{c \in C \mid \mathsf{src}(c) = p \vee \mathsf{dst}(c) = p\}$. The communication *type* of a process p on a channel $c \in C(p)$ that is not a self-loop, written $\mathsf{typ}(p, c)$, is the unique $* \in \{\bullet, \circ\}$ such that $(p, *)$ is an endpoint of c.

For each channel $c \in C$, we let $\overset{c}{\rightarrowtail}$ denote the binary relation on the set of processes P defined by $p \overset{c}{\rightarrowtail} q$ if $p = \mathsf{src}(c)$ and $q = \mathsf{dst}(c)$. Naturally, any topology may be viewed as the labeled *directed* graph $(P, \{\overset{c}{\rightarrowtail}\}_{c \in C})$. We assume some familiarity with classical notions on directed graphs, such as weak connectedness, strong connectedness, leaf nodes, etc. We also introduce the undirected binary relation $\overset{c}{\leftrightarrow}$, defined by $p \overset{c}{\leftrightarrow} q$ if $p \overset{c}{\rightarrowtail} q$ or $p \overset{c}{\leftarrowtail} q$. An *undirected path* in $\mathcal{T}$ is an alternating sequence $(p_0, c_1, p_1, \ldots, c_n, p_n)$, of processes $p_i \in P$ and channels $c_i \in C$, such that $p_{i-1} \overset{c_i}{\leftrightarrow} p_i$ for all $i \in \{1, \ldots, n\}$. Moreover, the undirected path is called *simple* when $p_0, \ldots, p_n$ are distinct. A *simple undirected cycle* in $\mathcal{T}$ is an undirected path $(p_0, c_1, p_1, \ldots, c_n, p_n)$, with $n \geq 1$, such that $p_1, \ldots, p_n$ are distinct, $c_1, \ldots, c_n$ are distinct, and $p_0 = p_n$. A *simple undirected shunt* in $\mathcal{T}$ is a simple undirected path $(p_0, c_1, p_1, \ldots, c_n, p_n)$, with $n \geq 2$, such that $\mathsf{typ}(p_0, c_1) = \bullet$ and $\mathsf{typ}(p_n, c_n) = \bullet$.

▶ **Definition 2.2.** Let $\mathcal{T}$ be a topology. $\mathcal{T}$ is called *cycle-free* if it contains no simple undirected cycle. $\mathcal{T}$ is called *shunt-free* if it contains no simple undirected shunt.

▶ Remark. Our notion of shunt is close to the *confluence* criterion presented in [13] for communicating pushdown processes. Simply put, confluence permits to synchronize two pushdown stacks, and a shunt permits to synchronize two counters, as will be seen later. However, shunts require at least one additional, intermediary process whereas confluence can be established directly between two processes. In our case, the topology $\mathtt{p} \overset{c}{\rightarrowtail} \mathtt{q}$ with channel endpoints of type $\bullet$ is shunt-free, and will be shown to have a decidable reachability problem.

Systems of Communicating One-Counter Machines. Classically, one-counter machines are finite-state automata, equipped with a counter, represented by a variable $\mathtt{x}$, that holds a non-negative integer value. The counter is initially set to zero, and can be incremented, decremented (provided that it remains non-negative), and tested for zero. In this paper, we consider an extension of counter machines where tests can be any unary Presburger predicate $\varphi(\mathtt{x})$. Such Presburger tests do not increase the expressive power of one-counter machines in terms of recognized languages [5]. We will show in the next section that the same property holds for their binary reachability relations. Presburger tests will be handy to merge several communicating one-counter machines in a single communicating one-counter machine.

Recall that *Presbuger arithmetic* is the first-order theory of the natural numbers with addition. A n-ary *Presburger predicate* is a Presburger formula φ with exactly n free variables. As usual, we write $\varphi(\mathtt{x}_1, \ldots, \mathtt{x}_n)$ to indicate that $\mathtt{x}_1, \ldots, \mathtt{x}_n$ are the free variables of φ. We let $\mathcal{P}_n$ denote the set of all n-ary Presburger predicates.

▶ **Definition 2.3.** A *system of communicating one-counter machines* is a pair $\mathcal{S} = \langle \mathcal{T}, (\mathcal{M}^p)_{p \in P} \rangle$ where $\mathcal{T}$ is a topology and, for each process p in P, $\mathcal{M}^p$ is a quintuple $\mathcal{M}^p = \langle S^p, I^p, F^p, A^p, \Delta^p \rangle$, called a *communicating one-counter machine*, where

- S^p is a finite set of *states*,
- $I^p, F^p \subseteq S^p$ are subsets of *initial states* and *final states*,
- $A^p \subseteq A_{\mathrm{cnt}} \cup A^p_{\mathrm{com}}$ is a finite set of *actions*, where
$$A_{\mathrm{cnt}} = \{\mathtt{add}(k) \mid k \in \mathbb{Z}\} \cup \{\mathtt{test}(\varphi) \mid \varphi \in \mathcal{P}_1\}$$
$$A^p_{\mathrm{com}} = \{c! \mid c \in C \wedge \mathsf{src}(c) = p\} \cup \{c? \mid c \in C \wedge \mathsf{dst}(c) = p\}$$
- $\Delta^p \subseteq S^p \times A^p \times S^p$ is a finite set of transition *rules*.

We give the operational semantics $[\![\mathcal{S}]\!]$ of a system of communicating one-counter machines $\mathcal{S}$ as a labeled transition system. A *configuration* of $[\![\mathcal{S}]\!]$ is triple $\sigma = (\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{w})$ where $\boldsymbol{s}$ maps each process p to a state in S^p, $\boldsymbol{x}$ maps each process p to a counter value in $\mathbb{N}$, and $\boldsymbol{w}$ maps each channel c to a word over the set of natural numbers. Formally, the set of

configurations of $[\![\mathcal{S}]\!]$ is $(\prod_{p\in P} S^p) \times \mathbb{N}^P \times (\mathbb{N}^*)^C$. An *initial configuration* is a configuration $(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{w})$ such that $\boldsymbol{x} = \mathbf{0}$, $\boldsymbol{w} = \boldsymbol{\varepsilon}$, and $s^p \in I^p$ for all $p \in P$. Analogously, a *final configuration* is a configuration $(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{w})$ such that $\boldsymbol{x} = \mathbf{0}$, $\boldsymbol{w} = \boldsymbol{\varepsilon}$, and $s^p \in F^p$ for all $p \in P$. The *transition relation* of $[\![\mathcal{S}]\!]$, written $\rightarrow$, is the set of all triples (σ_1, a, σ_2), where $\sigma_1 = (\boldsymbol{s}_1, \boldsymbol{x}_1, \boldsymbol{w}_1)$ and $\sigma_2 = (\boldsymbol{s}_2, \boldsymbol{x}_2, \boldsymbol{w}_2)$ are configurations, and a is an action in A^p, for some $p \in P$, satisfying the following conditions:

- $(s_1^p, a, s_2^p) \in \Delta^p$ and $s_1^q = s_2^q$ for all $q \in P$ with $q \neq p$,
- if $a = \mathtt{add}(k)$ then $x_2^p = x_1^p + k$, $x_1^q = x_2^q$ for all $q \in P$ with $q \neq p$, and $\boldsymbol{w}_1 = \boldsymbol{w}_2$,
- if $a = \mathtt{test}(\varphi(\mathtt{x}))$ then the valuation $\{\mathtt{x} \mapsto x_1^p\}$ satisfies $\varphi(\mathtt{x})$, $\boldsymbol{x}_1 = \boldsymbol{x}_2$ and $\boldsymbol{w}_1 = \boldsymbol{w}_2$,
- if $a = c!$, then
  - $w_2^c = w_1^c \cdot x_1^p$ and $w_1^d = w_2^d$ for all $d \in C$ with $d \neq c$, and
  - if $\mathsf{src}(c) = \bullet$ then $\boldsymbol{x}_1 = \boldsymbol{x}_2$; otherwise $x_1^q = x_2^q$ for all $q \in P$ with $q \neq p$.
- if $a = c?$, then
  - $w_1^c = x_2^p \cdot w_2^c$ and $w_1^d = w_2^d$ for all $d \in C$ with $d \neq c$, and
  - if $\mathsf{dst}(c) = \bullet$ then $\boldsymbol{x}_1 = \boldsymbol{x}_2$; otherwise $x_1^q = x_2^q$ for all $q \in P$ with $q \neq p$.

For readability, we write $\sigma_1 \xrightarrow{a} \sigma_2$ in place of $(\sigma_1, a, \sigma_2) \in \rightarrow$. Notice that we do not explicitly index actions by the process that fires them, but we assert that one implicitly knows which process moves on each transition. A *run* of $[\![\mathcal{S}]\!]$ is a finite, alternating sequence $\rho = (\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n)$ of configurations σ_i and actions a_i, satisfying $\sigma_{i-1} \xrightarrow{a_i} \sigma_i$ for all i. We say that ρ is a run *from* σ_0 *to* σ_n, and, abusing notation, we shortly write $\rho = \sigma_0 \xrightarrow{*} \sigma_n$. The *length* of ρ is n, and is denoted by $|\rho|$. A run of length zero consists of a single configuration. A *full run* of $[\![\mathcal{S}]\!]$ is a run from an initial configuration to a final configuration.

The semantics of counter operations $\mathtt{add}(k)$ and $\mathtt{test}(\varphi)$ is the usual one. A send or receive action on a channel appends or removes a message in $\mathbb{N}$, as one would expect. However, there are additional restrictions on the interplay of the communicated message and the local counter. If the endpoint of the channel has type $\bullet$, the message must equal the value of the counter *before* and *after* the action. So the value of the counter is not modified by a communication on this endpoint. On the contrary, if the endpoint has type $\circ$, then the local counter value is "lost" by a communication on this endpoint:

- an emission transfers the value of the counter from the process to the channel; the counter is non-deterministically set to an arbitrary value after the emission.
- a reception transfers the message from the channel to the local counter; the behavior mirrors that of an emission.

Exchange of Messages from a Finite Alphabet. On the contrary to classical *communicating finite-state machines* (CFSM), communicating one-counter machines cannot (directly) send or receive messages from an arbitrary finite alphabet M. However, they are able to perform these actions indirectly, as follows. Assume, without loss of generality, that M is a finite set of natural numbers. Sending a message $m \in M$ on a channel c, like a CFSM would, simply amounts to setting the local counter to m, and performing an emission on c. Receiving a message $m \in M$ from a channel c, like a CFSM would, is done by performing a reception from c, and checking that the received message is m. To realize this check, the machine

- simply sets its counter to m before the reception, for an endpoint with type $\bullet$,
- or checks that the counter equals m after the reception, for an endpoint with type $\circ$.

Note that in this simulation of CFSM-style communications, the counter is forcibly set to the (bounded) value corresponding to the message being exchanged, even for endpoints with type $\bullet$. We show, in the next section, another simulation of CFSM-style communications where one of the two peers is able to retain the value of its counter.

3 A Characterization of Topologies with Solvable Reachability

We investigate the power of systems of communicating one-counter machines with regard to their communication topology. Therefore, we introduce the reachability problem parametrized by a given topology. Recall that a full run of $[\![\mathcal{S}]\!]$ is a run from an initial configuration to a final configuration.

▶ **Definition 3.1.** Given a topology $\mathcal{T}$, the *reachability problem* for systems of communicating one-counter machines with topology $\mathcal{T}$, denoted by Rp-Sc1cm($\mathcal{T}$), is defined as follows:
Input: a system of communicating one-counter machines $\mathcal{S}$ with topology $\mathcal{T}$,
Output: whether there exists a full run in $[\![\mathcal{S}]\!]$.

The main result of the paper is a complete classification of the topologies that have a solvable reachability problem. We observe that, in absence of shunts, systems of communicating one-counter machines are still more expressive than Cfsm, but their reachability problems are decidable for the same topologies, namely, cycle-free topologies [17].

▶ **Theorem 3.2.** *Given a topology $\mathcal{T}$,* Rp-Sc1cm($\mathcal{T}$) *is decidable if and only if $\mathcal{T}$ is cycle-free and shunt-free.*

The proof of the theorem is presented at the end of this section for the "only if" direction, and in Section 4 for the "if" direction. Before that, let us provide a decomposition of topologies that are cycle-free and shunt-free. Observe that a weakly-connected topology is cycle-free if and only if there is a unique simple undirected path between every two processes.

▶ **Proposition 3.3.** *Let $\mathcal{T}$ be a weakly-connected topology with at least two processes. If $\mathcal{T}$ is cycle-free and shunt-free, then there are two distinct processes r, r', with $r \overset{c}{\leftrightarrow} r'$ for some channel c, such that, for every simple undirected path $(p_0, c_1, p_1, \ldots, c_n, p_n, d, q)$ with $p_0 \in \{r, r'\}$ and $q \notin \{r, r'\}$, the process q has type $\circ$ on the channel d.*

An example illustrating the proposition is provided in Figure 2(a). This weakly-connected topology is cycle-free and shunt-free. Therefore, its underlying undirected graph is a tree. The processes r and r' may be seen as two "roots", connected by a channel. All other processes are descendants of these two "roots", and have type $\circ$ on the channel (input or output) that leads to the root, as required by Proposition 3.3. Note, however, that r and r' are allowed to have type $\bullet$ on all channels. Recall that a process with type $\circ$ on a channel "loses" the value of its counter when it communicates over this channel. On the contrary, no loss of information occurs with type $\bullet$. But an endpoint with type $\bullet$ can simulate an endpoint with type $\circ$, by artificially "losing" the value of the local counter. We formalize this property by introducing the partial order $\sqsubseteq$ on $\{\circ, \bullet\}$ defined by $\circ \sqsubset \bullet$. This partial order is extended to endpoints in the natural way: $(p, *) \sqsubseteq (p', *')$ if $p = p'$ and $* \sqsubseteq *'$. Given two topologies $\mathcal{T} = \langle P_\mathcal{T}, C_\mathcal{T}, \mathsf{src}_\mathcal{T}, \mathsf{dst}_\mathcal{T} \rangle$ and $\mathcal{U} = \langle P_\mathcal{U}, C_\mathcal{U}, \mathsf{src}_\mathcal{U}, \mathsf{dst}_\mathcal{U} \rangle$, we say that $\mathcal{U}$ is a *sub-topology* of $\mathcal{T}$ if $P_\mathcal{U} \subseteq P_\mathcal{T}$, $C_\mathcal{U} \subseteq C_\mathcal{T}$, and, for every channel $c \in C_\mathcal{U}$, it holds that $\mathsf{src}_\mathcal{U}(c) \sqsubseteq \mathsf{src}_\mathcal{T}(c)$ and $\mathsf{dst}_\mathcal{U}(c) \sqsubseteq \mathsf{dst}_\mathcal{T}(c)$. As one would expect, sub-topologies have an easier reachability problem.

▶ **Proposition 3.4.** *For every topology $\mathcal{T}$ and for every sub-topology $\mathcal{U}$ of $\mathcal{T}$,* Rp-Sc1cm($\mathcal{U}$) *is reducible to* Rp-Sc1cm($\mathcal{T}$).

Cycle-freeness and Shunt-freeness of Decidable Topologies. In the remainder of this section, we prove the "only if" direction of Theorem 3.2, namely that Rp-Sc1cm($\mathcal{T}$) is undecidable if $\mathcal{T}$ contains a simple undirected cycle or a simple undirected shunt. As seen in Section 2, systems of communicating one-counter machines can simulate Cfsm, and

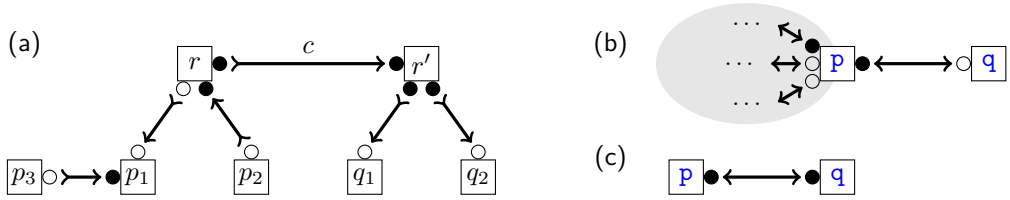


Figure 2 Topologies: (a) weakly connected cycle-free and shunt-free topology, (b) topology containing a leaf process q with type ∘ on its pendant channel, (c) decidable two-processes case.

the simulation preserves the topology. Moreover, the reachability problem for CFSM with topology $\mathcal{T}$ is known to be undecidable if $\mathcal{T}$ contains a simple undirected cycle [17, 14]. It follows that RP-SC1CM($\mathcal{T}$) is undecidable if $\mathcal{T}$ contains a simple undirected cycle. The following lemma completes the proof of the "only if" direction of Theorem 3.2.

▶ **Lemma 3.5.** *For every topology $\mathcal{T}$ containing a simple undirected shunt,* RP-SC1CM($\mathcal{T}$) *is undecidable.*

We explain the main ideas of the proof on the topology $\mathtt{p} \overset{\mathtt{c}}{\rightarrowtail} \mathtt{r} \overset{\mathtt{d}}{\leftarrowtail} \mathtt{q}$ where r has type ∘ on channels c and d, p has type • on c and q has type • on d. Let us call this topology $\mathcal{T}$. Notice that (p, c, r, d, q) is a simple undirected shunt. We show that the reachability problem for two-counters (Minsky) machines, which is known to be undecidable [16], is reducible to RP-SC1CM($\mathcal{T}$). Given a two-counters machine $\mathcal{M}$, one counter, say x, is maintained by p, and the other, say y, is maintained by q. Both processes p and q run a copy of $\mathcal{M}$, but they internalize (as add(0) actions) the counter actions of $\mathcal{M}$ that do not involve their counter. We only need to make sure that p and q take the same control path of $\mathcal{M}$. To this end, p and q send to r the transition rules that they traverse, and r checks that these rules are the same. However, p and q must not lose the value of their counter when communicating with r. So the simulation of CFSM presented in Section 2 cannot be used. Instead, p and q encode the transition rules within the counter value itself, send it to r, and let r decode and check this information.

Assume that $\mathcal{M}$ contains $K > 0$ transition rules, encoded as $0, \ldots, K-1$. Instead of storing the values x and y of x and y in their local counters, p and q store $K \cdot x$ and $K \cdot y$, respectively. So, increments and decrements in $\mathcal{M}$ are multiplied by the constant K in p and q. On the sender side, when p or q takes a transition rule encoded by $\delta \in \{0, \ldots, K-1\}$, it increments its counter by δ, sends it to r, and decrements its counter by δ to restore its value. On the receiver side, when r performs a c ? action, its counter is set to the message $m = \delta + (K \cdot x)$ sent by p, and r extracts the transition rule δ by computing (m mod K). The transition rules taken by q are decoded by r similarly.

The simulation guarantees that the two-counters machine has a full run if and only if the constructed system of communicating one-counter machines, with topology $\mathcal{T}$, has a full run. It follows that RP-SC1CM($\mathcal{T}$) is undecidable. Note that, by Proposition 3.4, the reachability problem RP-SC1CM($\mathcal{T}$) would also be undecidable (and even more so) if r had type • instead of ∘ on its output channels.

▶ Remark. We need at least one intermediary process r between p and q, to decode and check their messages. Indeed, direct communications between p and q would synchronize their local counters, thus making it impossible to maintain two counters.

4 Decidability of Cycle-free and Shunt-free Topologies

This section is devoted to the proof of the "if" direction of Theorem 3.2, namely that RP-SC1CM($\mathcal{T}$) is decidable if $\mathcal{T}$ is cycle-free and shunt-free. Without loss generality, we only consider weakly-connected topologies. The proof comprises three independent parts. Firstly, we provide a characterization, in terms of Presburger predicates, of reachability relations of one-counter machines. Secondly, we show that any leaf process with type $\circ$ on its pendant channel may be merged into its parent, thereby reducing the size of the topology. Iterating this reduction leads to a topology with only two processes and one channel. We show, in the third part, that RP-SC1CM($\mathcal{T}$) is decidable for such topologies.

Counter reachability relations of one-counter machines. A *one-counter machine* is a communicating one-counter machine $\mathcal{M} = \langle S, I, F, A, \Delta \rangle$ with no communication action, i.e., $A \subseteq A_{\text{cnt}}$. To fit our framework, we identify $\mathcal{M}$ with the system $\langle \mathcal{U}, (\mathcal{M}^p)_{p \in \{\mathtt{p}\}} \rangle$ of communicating one-counter machines, where $\mathcal{U} = \langle \{\mathtt{p}\}, \emptyset, \mathsf{src}, \mathsf{dst} \rangle$ is the topology with a single process $\mathtt{p}$ and no channel. We let RP-1CM denote the reachability problem for one-counter machines, formally RP-1CM = RP-SC1CM($\mathcal{U}$). It is well-known that RP-1CM is decidable since reachability is decidable for the more general class of pushdown systems.

In the next subsections, we show that, under certain conditions, two processes can be merged in a single "product" process (with only one counter). To do so, we summarize the behavior of a process between each communication action. This subsection is devoted to the characterization and computation of these summaries.

Let $\mathcal{M} = \langle S, I, F, A, \Delta \rangle$ be a one-counter machine. The *counter reachability relation* of $\mathcal{M}$ is the set of all pairs $(x, y) \in \mathbb{N} \times \mathbb{N}$ such that, for some $s \in I$ and $t \in F$, there exists a run from (s, x) to (t, y). To characterize counter reachability relations, we introduce the following class of binary Presburger predicates. We consider two distinguished Presburger variables $\mathtt{x}$ and $\mathtt{y}$. In short, one-counter Presburger predicates can express properties of $\mathtt{x}$, of $\mathtt{y}$, and of their differences $\mathtt{x} - \mathtt{y}$ and $\mathtt{y} - \mathtt{x}$. Formally, the class of *one-counter Presburger predicates* is generated by the grammar:

$$\psi \quad ::= \quad \varphi(\mathtt{x}) \mid \varphi(\mathtt{y}) \mid \exists z \cdot (\mathtt{x} = \mathtt{y} + z \wedge \varphi(z)) \mid \exists z \cdot (\mathtt{y} = \mathtt{x} + z \wedge \varphi(z)) \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathtt{tt} \mid \mathtt{ff}$$

where φ ranges over the set $\mathcal{P}_1$ of unary Presburger predicates. The binary relation *defined* by a one-counter Presburger predicate ψ is the set of all pairs $(x, y) \in \mathbb{N} \times \mathbb{N}$ such that the valuation $\{\mathtt{x} \mapsto x, \mathtt{y} \mapsto y\}$ satisfies ψ.

We first show that counter reachability relations are definable by one-counter Presburger predicates, for the class of one-counter machines with zero-tests only. Formally, a one-counter machine $\mathcal{M} = \langle S, I, F, A, \Delta \rangle$ is called *basic* if $A \subseteq \{\mathtt{add}(k) \mid k \in \mathbb{Z}\} \cup \{\mathtt{test}(\mathtt{x} = 0)\}$.

▶ **Lemma 4.1.** *For every basic one-counter machine $\mathcal{M}$, the counter reachability relation of $\mathcal{M}$ is defined by a one-counter Presburger predicate.*

However, the converse of the lemma does not hold. Consider, for instance, the one-counter Presburger predicate $\psi = \exists \mathtt{k} \cdot (\mathtt{x} = \mathtt{k} + \mathtt{k}) \wedge (\mathtt{x} = \mathtt{y})$. In a basic one-counter machine, it is not possible to check that a given, a priori unknown value $\mathtt{x}$ is even without "losing" this value. We need the additional expressive power stemming from Presburger tests.

We now show that counter reachability relations (of arbitrary one-counter machines) are precisely the relations definable by one-counter Presburger predicates. This entails, in particular, that counter reachability relations are closed under intersection. We will use this

property in the proof of Lemma 4.4. On the logical side, we obtain that the class of relations definable by one-counter Presburger predicates is closed under transitive closure.

▶ **Theorem 4.2.** *For every binary relation $R \subseteq \mathbb{N} \times \mathbb{N}$, the two following assertions are equivalent:*

- *R is the counter reachability relation of a one-counter machine,*
- *R is defined by a one-counter Presburger predicate.*

▶ Remark. The proof of Theorem 4.2 is constructive, in the sense that a one-counter Presburger predicate is computable from a given one-counter machine, and vice versa.

Merging leaf processes. We show how to reduce the number of processes in a system of communicating one-counter machines, by merging a leaf process with type $\circ$ on its pendant channel into its parent. Let $\mathcal{U} = \langle P_{\mathcal{U}}, C_{\mathcal{U}}, \mathsf{src}_{\mathcal{U}}, \mathsf{dst}_{\mathcal{U}} \rangle$ be a topology, and select a distinguished process $\mathtt{p}$ in $P_{\mathcal{U}}$. We add to the topology a new process $\mathtt{q} \notin P_{\mathcal{U}}$ and a new channel $\mathtt{c} \notin C_{\mathcal{U}}$ between $\mathtt{p}$ and $\mathtt{q}$. Formally, we consider any topology $\mathcal{T} = \langle P, C, \mathsf{src}, \mathsf{dst} \rangle$ with set of processes $P = P_{\mathcal{U}} \cup \{\mathtt{q}\}$ and set of channels $C = C_{\mathcal{U}} \cup \{\mathtt{c}\}$, whose source and destination mappings coincide with those of $\mathcal{U}$ on C, and such that $\mathtt{p} \overset{\mathtt{c}}{\leftrightarrow} \mathtt{q}$. Observe that $C(\mathtt{q}) = \{\mathtt{c}\}$, hence, $\mathtt{q}$ is a leaf process with pendant channel $\mathtt{c}$. The topology $\mathcal{T}$ is depicted on Figure 2(b).

▶ **Lemma 4.3.** *If $\mathtt{p}$ has type $\bullet$ on $\mathtt{c}$ and $\mathtt{q}$ has type $\circ$ on $\mathtt{c}$ then* Rp-Sc1cm$(\mathcal{T})$ *is reducible to* Rp-Sc1cm$(\mathcal{U})$.

Let us explain the main ideas of the proof. Assume that $\mathtt{c}$ is directed as $\mathtt{p} \overset{\mathtt{c}}{\rightarrowtail} \mathtt{q}$. Consider a system of communicating one-counter machines $\mathcal{S} = \langle \mathcal{T}, (\mathcal{M}^p)_{p \in P} \rangle$. To simulate $\mathcal{S}$ over the topology $\mathcal{U}$, we merge processes $\mathtt{p}$ and $\mathtt{q}$ in a single "product" process $\widehat{\mathtt{p}}$. So, the communicating one-counter machines $\mathcal{M}^p$ are kept unchanged for all processes in $p \in P \setminus \{\mathtt{p}, \mathtt{q}\}$. But the process $\widehat{\mathtt{p}}$ must simulate both processes $\mathtt{p}$ and $\mathtt{q}$, as well as the channel $\mathtt{c}$ in-between. We choose a specific interleaving of $\mathtt{p}$ and $\mathtt{q}$ where $\mathtt{c}$ is almost always empty, and such that $\widehat{\mathtt{p}}$, which has a single counter, is able to retain both $\mathtt{p}$'s counter and $\mathtt{q}$'s counter.

In essence, $\widehat{\mathtt{p}}$ behaves as $\mathtt{p}$, but also maintains, in its state, the local state of $\mathtt{q}$ as well as an abstraction of $\mathtt{q}$'s counter. We abstract $\mathtt{q}$'s counter by the set $\{0, \bot, =\}$, where 0 means zero, $\bot$ means some unknown value, and $=$ means that $\mathtt{q}$'s counter holds the same value as $\mathtt{p}$'s counter. Furthermore, the process $\mathtt{q}$ is always scheduled first. Since $\mathtt{c}$ is the only channel with source or destination $\mathtt{q}$, this means, in particular, that every reception by $\mathtt{q}$ from $\mathtt{c}$ occurs immediately after the matching emission by $\mathtt{p}$ on $\mathtt{c}$. When $\widehat{\mathtt{p}}$ simulates an emission by $\mathtt{p}$ on $\mathtt{c}$ and the matching reception by $\mathtt{q}$, it internalizes this synchronization $\mathtt{c}! \cdot \mathtt{c}?$, and sets $\mathtt{q}$'s abstract counter to $=$. Indeed, since $\mathtt{q}$ has type $\circ$ on $\mathtt{c}$, the reception by $\mathtt{q}$ from $\mathtt{c}$ overwrites its counter with the value of $\mathtt{p}$'s counter. Then, $\widehat{\mathtt{p}}$ simulates, in one step, the behavior of $\mathtt{q}$ from this matching reception to the next reception. Observe that the next reception of $\mathtt{q}$ from $\mathtt{c}$ will, again, overwrite its counter. Therefore, thanks to Theorem 4.2, this behavior of $\mathtt{q}$ can be summarized in a single Presburger test, that accounts for the local state reached by $\mathtt{q}$. This way, $\widehat{\mathtt{p}}$ does not need to maintain the value held by $\mathtt{q}$'s counter. The construction guarantees that $\mathcal{S}$ has a full run if and only if the resulting system of communicating one-counter machines, with topology $\mathcal{U}$, has a full run.

The proof for the other direction $\mathtt{q} \overset{\mathtt{c}}{\rightarrowtail} \mathtt{p}$ is similar. However, instead of scheduling $\mathtt{q}$ first, it is now scheduled last.

Two processes connected by one channel. We now consider the topology depicted on Figure 2(c), with two distinct processes $\mathtt{p}$ and $\mathtt{q}$ and a channel from $\mathtt{p}$ to $\mathtt{q}$ with type $\bullet$ on both endpoints. Formally, $\mathcal{T} = \langle \{\mathtt{p}, \mathtt{q}\}, \{\mathtt{c}\}, \mathsf{src}, \mathsf{dst} \rangle$ with $\mathsf{src}(\mathtt{c}) = (\mathtt{p}, \bullet)$ and $\mathsf{dst}(\mathtt{c}) = (\mathtt{q}, \bullet)$.

▶ **Lemma 4.4.** RP-SC1CM($\mathcal{T}$) *is reducible to* RP-1CM.

Informally, given a system of communicating one-counter machines $\mathcal{S} = \langle \mathcal{T}, (\mathcal{M}^p)_{p \in P} \rangle$, we construct a one-counter machine $\mathcal{N}$ that simulates the "product" of p and q. As in the proof of Lemma 4.3, we schedule the sender last (here, p) and the receiver first (here, q). Thus, emissions c! and receptions c? occur consecutively, with no other action in between. Since p and q have type $\bullet$ on c, each sequence of actions c! · c? may occur only if p's counter and q's counter hold the same value. So $\mathcal{N}$ internalizes each synchronization c! · c?, and simulates, in one step, the behavior of p and q from one synchronization to the next. This is possible thanks to Theorem 4.2, which entails that counter reachability relations are (effectively) closed under intersection. The construction guarantees that $\mathcal{S}$ has a full run if and only if the constructed one-counter machine $\mathcal{N}$ has a full run.

Wrap up. We now have the necessary ingredients to prove the "if" direction of Theorem 3.2. Consider a weakly-connected topology $\mathcal{T}$ that is both cycle-free and shunt-free. We show that RP-SC1CM($\mathcal{T}$) is reducible to RP-1CM. If $\mathcal{T}$ contains only one process, then $\mathcal{T}$ contains no channel as it is cycle-free, hence, RP-SC1CM($\mathcal{T}$) is obviously reducible to RP-1CM. Assume that $\mathcal{T}$ contains at least two processes. By Proposition 3.3, there exists two distinct processes r, r' and a channel c, with $r \overset{c}{\leftrightarrow} r'$, such that, for every simple undirected path $(p_0, c_1, p_1, \ldots, c_n, p_n, d, q)$ with $p_0 \in \{r, r'\}$ and $q \notin \{r, r'\}$, the process q has type $\circ$ on the channel d. Moreover, according to Proposition 3.4, we may replace some endpoints $(p, \circ)$ by $(p, \bullet)$, as the reachability problem RP-SC1CM($\mathcal{T}$) is reducible to the reachability problem for the transformed topology. So we assume, without loss of generality, that for every simple undirected path $(p_0, c_1, p_1, \ldots, c_n, p_n, p, d, q)$ with $p_0 \in \{r, r'\}$, the process p has type $\bullet$ on the channel d. In particular, r and r' have type $\bullet$ on c.

Since $\mathcal{T}$ is cycle-free, its underlying undirected graph $(P, \{\overset{c}{\leftrightarrow}\}_{c \in C})$ is a tree. Pick a leaf process q that is distinct from r and r' (if any). Let $\mathcal{T} - q$ denote the topology obtained from $\mathcal{T}$ by removing the process q as well as its pendant channel. The simple undirected path from r to q ends with a channel $p \overset{d}{\leftrightarrow} q$ that satisfies $C(q) = \{d\}$, p has type $\bullet$ on d and q has type $\circ$ on d. It follows from Lemma 4.3 that RP-SC1CM($\mathcal{T}$) is reducible to RP-SC1CM($\mathcal{T} - q$). By iterating this elimination technique in a bottom-up fashion, we obtain that RP-SC1CM($\mathcal{T}$) is reducible to RP-SC1CM($\mathcal{U}$) where $\mathcal{U}$ is the topology consisting of the two processes r, r' and the single channel c. According to Lemma 4.4, RP-SC1CM($\mathcal{U}$) is reducible to RP-1CM. We conclude that RP-SC1CM($\mathcal{T}$) is reducible to RP-1CM. Since the latter decidable, we get that the former is decidable, too.

5 Systems with Eager Communication

As seen in our motivating example of Figure 1, cyclic topologies are the backbone of communication protocols. However, already for CFSM, the reachability problem is undecidable in presence of cycles, which is also mirrored in Theorem 3.2. In this section, we consider a restriction to so-called eager runs. This restriction provides an under-approximative answer to the reachability problem RP-SC1CM($\mathcal{T}$) considered in the previous sections. Eager runs are close to globally 1-bounded runs, and have been successfully applied, in combination with other restrictions, to the reachability analysis of communicating pushdown processes [13].

▶ **Definition 5.1.** A full run $\rho = (\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n)$ in $[\![\mathcal{S}]\!]$ is called *eager* if, for every channel c and for every index $i \in \{1, \ldots, n-1\}$, it holds that $a_i = c!$ if and only if $a_{i+1} = c?$.

Thus, eagerness transforms asynchronous message-passing communications into rendezvous synchronizations. This may seem rather restrictive. Actually, eagerness is equivalent, up to re-ordering[1], to the requirement that all other channels be empty when one channel is transferring a message [13]. Therefore, eagerness encompasses half-duplex communication.

The *eager-reachability problem* RP-SC1CM-EAGER($\mathcal{T}$) is defined in the same way as RP-SC1CM($\mathcal{T}$) except that we search for a full run that must be eager. By definition, this problem provides an under-approximative answer to RP-SC1CM($\mathcal{T}$). This under-approximation is exact when the topology is cycle-free. Indeed, for such topologies, full runs can be re-ordered into eager ones [13]. It follows from Theorem 3.2 that, for every cycle-free topology $\mathcal{T}$, RP-SC1CM-EAGER($\mathcal{T}$) is decidable if and only if $\mathcal{T}$ is shunt-free. Hence, eagerness is only interesting in presence of cycles. For the remainder of this section, we focus on cyclic communication. The following proposition establishes the decidability frontier of the eager-reachability problem for the particular case of strongly-connected topologies.

▶ **Proposition 5.2.** *Given a strongly-connected topology $\mathcal{T}$,* RP-SC1CM-EAGER($\mathcal{T}$) *is decidable if and only if $\mathcal{T}$ contains at most two processes.*

We first consider the simplest strongly-connected topology with two processes $\mathtt{p} \overset{\mathtt{c}}{\rightarrowtail} \mathtt{q} \overset{\mathtt{d}}{\rightarrowtail} \mathtt{p}$, where all channel endpoints have type $\bullet$. Then, eagerness allows us to reverse the direction of a channel, leading to $\mathtt{p} \overset{\mathtt{c}}{\rightarrowtail} \mathtt{q} \overset{\mathtt{d}}{\leftarrowtail} \mathtt{p}$. With the same encoding as in Lemma 3.5, we may tag each message by the channel c or d that it is sent over. As eager message passing only uses one channel at a time, we can assert that all messages are now passed over one common channel. Hence we can apply the decidability result of Lemma 4.4 on two processes connected by one channel. This construction can be extended to more than two channels between p and q. A strongly-connected topology may also contain self-loops, but they become irrelevant by the restriction to eager runs. Finally, we extend this result to topologies with channel endpoints of type $\circ$ by Proposition 3.4 (generalized to eager-reachability).

For the converse, consider a strongly-connected component with at least three processes. We may assume, without loss generality, that all channel endpoints have type $\circ$. The component necessarily contains (a) a directed cycle of length at least three, i.e., assuming for simplicity that the length is three, a sub-topology $\mathcal{T}_a$ of the form $\mathtt{p} \overset{\mathtt{c}}{\rightarrowtail} \mathtt{q} \overset{\mathtt{d}}{\rightarrowtail} \mathtt{r} \overset{\mathtt{e}}{\rightarrowtail} \mathtt{p}$, or ($b$) two directed cycles, each of length two, that are disjoint except for one common process, i.e., a sub-topology $\mathcal{T}_b$ of the form $\mathtt{q} \overset{\mathtt{c}}{\rightarrowtail} \mathtt{p} \overset{\mathtt{d}}{\rightarrowtail} \mathtt{r} \overset{\mathtt{e}}{\rightarrowtail} \mathtt{p} \overset{\mathtt{f}}{\rightarrowtail} \mathtt{q}$. We show a reduction from the reachability problem for two-counters machines. The restriction to eager runs guarantees that each send is immediately followed by the matching receive. We use this restriction to implement a protocol that gives one distinguished process access to the two counters, the latter being stored and passed around in the topology without getting lost. In the case of $\mathcal{T}_a$, process p simulates the two-counters machine by maintaining one of the counters locally, and the other at r. To let p use the other counter, the protocol ensures that we switch the counters by using q as buffer. In the case of $\mathcal{T}_b$, the two-counters machine is simulated by p, while q and r are used as registers for either one of the two counters.

Let us come back to the sliding window protocol of Figure 1. Assume that, in both processes, receptions have precedence over transmissions. This priority ensures that channels are used in a half-duplex way. By [13], every full run can then be re-ordered into an eager one. Since the topology of Figure 1 falls in the scope of the previous proposition, we can decide whether the protocol is safe or not (when priority is given to receptions).

[1] A run ρ can be *re-ordered* into a run ρ' if ρ can be transformed into ρ' by iteratively commuting adjacent transitions that (i) are from different processes, and (ii) do *not* form a matching send/receive pair.

6 Conclusion and Perspectives

Systems of communicating one-counter machines introduce two additional sources of infinity with respect to CFSM, namely, the infinite message alphabet and the local counters. Thanks to a characterization of one-counter reachability relations in terms of binary Presburger predicates, we have obtained a complete classification of the topologies having a solvable reachability question. This shows, in particular, that decidable topologies are the same as for the weaker model of CFSM (provided that they contain no shunt). To address topologies allowing mutual communications, we have considered an under-approximative approach by restricting runs to eager ones. As a preliminary result, we have characterized the strongly-connected topologies that have a solvable eager-reachability question. A complete characterization of decidable topologies for eager reachability is currently under investigation. Further, we plan to extend our results from counters to stacks, i.e., to systems of communicating pushdown machines that can exchange the value of their stacks.

References

1 P. Abdulla, B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
2 S. Böhm, S. Göller, P. Jančar. Bisimilarity of one-counter processes is PSPACE-complete. In *Proc. CONCUR'10*, *LNCS* 6269, pp. 177–191. Springer, 2010.
3 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on data words. *ACM Trans. Computational Logic*, 12(4):27, 2011.
4 B. Bollig, A. Cyriac, P. Gastin, K. Narayan Kumar. Model checking languages of data words. In *Proc. FOSSACSS'12*, *LNCS* 7213, pp. 391–405. Springer, 2012.
5 A. Bouajjani, P. Habermehl, R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.
6 D. Brand, P. Zafiropoulo. On communicating finite-state machines. Research Report 1053, IBM Zürich Research Laboratory, 1981.
7 P. Chambart, P. Schnoebelen. Mixing lossy and perfect fifo channels. In *Proc. CONCUR'08*, *LNCS* 5201, pp. 340–355, 2008.
8 S. Demri, R. Lazic, A. Sangnier. Model checking freeze LTL over one-counter automata. In *Proc. FOSSACS'08*, *LNCS* 4962, pp. 490–504. Springer, 2008.
9 A. Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3):129–135, 1994.
10 A. Finkel, G. Memmi. Fifo nets: a new model of parallel computation. In *Proc. TCS'83*, *LNCS* 145, pp. 111–121. Springer, 1983.
11 A. Finkel, G. Sutre. Decidability of reachability problems for classes of two counters automata. In *Proc. STACS'00*, *LNCS* 1770, pp. 346–357. Springer, 2000.
12 S. Göller, C. Haase, J. Ouaknine, J. Worrell. Branching-time model checking of parametric one-counter automata. In *Proc. FOSSACS'12*, *LNCS* 7213, pp. 406–420. Springer, 2012.
13 A. Heußner, J. Leroux, A. Muscholl, G. Sutre. Reachability analysis of communicating pushdown systems. *Logical Methods in Computer Science*, 8(3:23):1–20, 2012.
14 S. La Torre, P. Madhusudan, G. Parlato. Context-bounded analysis of concurrent queue systems. In *Proc. TACAS'08*, *LNCS* 4963, pp. 299–314. Springer, 2008.
15 T. Le Gall, B. Jeannet. Lattice automata In *Proc. SAS'07*, *LNCS* 4634, pp. 52–68. Springer, 2007.
16 M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
17 J.K. Pachl. Reachability problems for communicating finite state machines. Research Report CS-82-11, Dept. of C.S. Univ. of Waterloo, 1982.

Density Functions subject to a Co-Matroid Constraint*

Venkatesan T. Chakaravarthy[1], Natwar Modani[1], Sivaramakrishnan R. Natarajan[2], Sambuddha Roy[1], and Yogish Sabharwal[1]

1 **IBM Research, New Delhi, India**
{vechakra,namodani,sambuddha,ysabharwal}@in.ibm.com
2 **IIT Chennai, India**
sivaramakrishnan.n.r@gmail.com

Abstract

In this paper we consider the problem of finding the *densest* subset subject to *co-matroid constraints*. We are given a *monotone supermodular* set function f defined over a universe U, and the density of a subset S is defined to be $f(S)/|S|$. This generalizes the concept of graph density. Co-matroid constraints are the following: given matroid $\mathcal{M}$ a set S is feasible, iff the complement of S is *independent* in the matroid. Under such constraints, the problem becomes NP-hard. The specific case of graph density has been considered in literature under specific co-matroid constraints, for example, the cardinality matroid and the partition matroid. We show a 2-approximation for finding the densest subset subject to co-matroid constraints. Thereby we improve the approximation guarantees for the result for partition matroids in the literature.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, Submodular Functions, Graph Density

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.236

1 Introduction

In this paper, we consider the problem of computing the densest subset with respect to a *monotone supermodular* function subject to *co-matroid* constraints. Given a universe U of n elements, a function $f : 2^U \to \mathbb{R}^+$ is *supermodular* iff

$$f(A) + f(B) \leqslant f(A \cup B) + f(A \cap B)$$

for all $A, B \subseteq U$. If the sign of the inequality is reversed for all A, B, then we call the function *submodular*. The function f is said to be monotone if $f(A) \leqslant f(B)$ whenever $A \subseteq B$; we assume $f(\emptyset) = 0$. We define a *density function* $d : 2^U \to \mathbb{R}^+$ as $d(S) \triangleq f(S)/|S|$. Consider the problem of maximizing the density function $d(S)$ given oracle access to the function f. We observe that the above problem can be solved in polynomial time (see Theorem 6).

The main problem considered in this paper is to maximize $d(S)$ subject to certain constraints that we call *co-matroid* constraints. In this scenario, we are given a *matroid* $\mathcal{M} = (U, \mathcal{I})$ where $\mathcal{I} \subseteq 2^U$ is the family of *independent* sets (we give the formal definition of a matroid in Section 2). A set S is considered feasible iff the complement of S is *independent*

* Work done by the third author while he was interning at IBM Research

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 236–248

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

i.e. $\overline{S} \in \mathcal{I}$. The problem is to find the densest feasible subset S given oracle access to f and $\mathcal{M}$. We denote this problem as DEN-M.

We note that even special cases of the DEN-M problem are NP-hard [14]. The main result in this paper is the following:

▶ **Theorem 1.** *Given a monotone supermodular function f over a universe U, and a matroid $\mathcal{M}$ defined over the same universe, there is a 2-approximation algorithm for the* DEN-M *problem.*

Alternatively one could have considered the same problem under *matroid constraints* (instead of co-matroid constraints). We note that this problem is significantly harder, since the Densest Subgraph problem can be reduced to special cases of this problem (see [2, 14]). The Densest Subgraph problem is notoriously hard: the best factor approximation known to date is $O(n^{1/4+\epsilon})$ for any $\epsilon > 0$ [3].

Special cases of the DEN-M problem have been extensively studied in the context of graph density, and we discuss this next.

1.1 Comparison to Graph Density

Given an undirected graph $G = (V, E)$, the density $d(S)$ of a subgraph on vertex set S is defined as the quantity $\frac{|E(S)|}{|S|}$, where $E(S)$ is the set of edges in the subgraph induced by the vertex set S. The densest subgraph problem is to find the subgraph S of G that maximizes the density.

The concept of graph density is ubiquitous, more so in the context of social networks. In the context of social networks, the problem is to detect *communities*: collections of individuals who are relatively well connected as compared to other parts of the social network graph.

The results relating to graph density have been fruitfully applied to finding communities in the social network graph (or even web graphs, gene annotation graphs [15], problems related to the formation of most effective teams [9], etc.). Also, note that graph density appears naturally in the study of threshold phenomena in random graphs, see [1].

Motivated by applications in social networks, the graph density problem and its variants have been well studied. Goldberg [11] proved that the densest subgraph problem can be solved optimally in polynomial time: he showed this via a reduction to a series of max-flow computations. Later, others [7, 14] have given new proofs for the above result, motivated by considerations to extend the result to some generalizations and variants.

Andersen and Chellapilla [2] studied the following generalization of the above problem. Here, the input also includes an integer k, and the goal is to find the densest subgraph S subject to the constraint $|S| \geqslant k$. This corresponds to finding *sufficiently large* dense subgraphs in social networks. This problem is NP-hard [14]. Andersen and Chellapilla [2] gave a 2-approximation algorithm. Khuller and Saha [14] give two alternative algorithms: one of them is a greedy procedure, while the other is LP-based. Both the algorithms have 2-factor guarantees.

Gajewar and Sarma [9] consider a further generalization. The input also includes a partition of the vertex set into $U_1, U_2, \cdots, U_t$, and non-negative integers $r_1, r_2, \cdots, r_t$. The goal is to find the densest subgraph S subject to the constraint that for all $1 \leqslant i \leqslant t$, $|S \cap U_i| \geqslant r_i$. They gave a 3-approximation algorithm by extending the greedy procedure of Khuller and Saha [14].

We make the following observations: (i) The objective function $|E(S)|$ is monotone and supermodular. (ii) The constraint $|S| \geqslant k$ (considered by [2]) is a co-matroid constraint; this corresponds to the *cardinality matroid*. (iii) The constraint considered by Gajewar and

Sarma [9] is also a co-matroid constraint; this corresponds to the *partition matroid* (formal definitions are provided in Section 2). Consequently, our main result Theorem 1 improves upon the above results in three directions:

- *Objective function*: Our results apply to general monotone supermodular functions f instead of the specific set function $|E(S)|$ in graphs.
- *Constraints*: We allow co-matroid constraints corresponding to *arbitrary* matroids.
- *Approximation Factor:* For the problem considered by Gajewar and Sarma [9], we improve the approximation guarantee from 3 to 2. We match the best factor known for the at-least-k densest subgraph problem considered in [2, 14].

1.2 Other Results

Knapsack Covering Constraints:

We also consider the following variant of the `DEN-M` problem. In this variant, we will have a weight w_i (for $i = 1, \cdots, |U|$) for every element $i \in U$, and a number $k \in \mathbb{N}$. A set S of elements is *feasible* if and only if the following condition holds: $\sum_{i \in S} w_i \geqslant k$. We call this a *knapsack covering constraint*. We extend the proof of Theorem 1 to show the following:

▶ **Theorem 2.** *Suppose we are given a monotone supermodular function f over a universe U, weights w_i for every element $i \in U$, and a number $k \in \mathbb{N}$. Then there is a* 3*-approximation algorithm for maximizing the* density function $d(S)$ *subject to* knapsack covering constraints *corresponding to the weights w_i and the number k.*

Dependency Constraints:

Saha et. al[15] consider a variant of the graph density problem. In this version, we are given a specific collection of vertices $A \subseteq V$; a subset S of vertices is *feasible* iff $A \subseteq S$. We call this restriction the *subset* constraint. The objective is to find the densest subgraph among subsets satisfying a subset constraint. Saha et. al[15] prove that this problem is solvable in polynomial time by reducing this problem to a series of max-flow computations.

We study a generalization of the subset constraint problem. Here, we are given a monotone supermodular function f defined over universe U. Additionally, we are given a *directed graph* $D = (U, \vec{A})$ over the universe U. A feasible solution S has to satisfy the following property: if $a \in S$, then every vertex of the digraph D *reachable* from a also has to belong to S. Alternatively, $a \in S$ and $(a, b) \in \vec{A}$ implies that $b \in S$. We call the digraph D as the *dependency graph* and such constraints as *dependency* constraints. The goal is to find the densest subset S subject to the dependency constraints. We call this the `DENdep` problem. We note that the concept of dependency constraints generalizes that of the subset constraints: construct a digraph D by drawing directed arcs from every vertex in U to every vertex in A. The motivation for this problem comes from certain considerations in social networks, where we are to find the densest subgraph but with the restriction that in the solution subgraph all the members of a sub-community (say, a family) are present or absent simultaneously. In literature, such a solution S that satisfies the dependency constraints is also called a *closure* (see [18], Section 3.7.2). Thus our problem can be rephrased as that of finding the densest subset over all closures.

We note that dependency constraints are incomparable with co-matroid constraints. In fact dependency constraints are not even upward monotone: it is *not* true that if S is a feasible subset, *any* superset of S is feasible.

Our result is as follows:

▶ **Theorem 3.** *The* `DENdep` *problem is solvable in polynomial time.*

The salient features of the above result are as follows:

- While the result in [15] is specific to graph density, our result holds for density functions arising from arbitrary monotone supermodular functions.
- Our proof of this result is LP-based. The work of [15] is based on max-flow computations. We can extend our LP-based approach (via convex programs) to the case for density functions arising from arbitrary monotone supermodular f, while we are not aware as to how to extend the max-flow based computation.
- The proof technique, inspired by Iwata and Nagano [13] also extends to show "small support" results: thus, for instance, we can show that for the LP considered by [14] for the at-least-k-densest subgraph problem, every *non-zero* component of any basic feasible solution is one of *two* values.

Combination of Constraints:

We also explore the problem of finding the densest subset subject to a combination of the constraints considered. We are able to prove results for the problem of maximizing a density function subject to (a) *co-matroid* constraints and (b) *subset* constraints. Suppose we are given a monotone supermodular function f over a universe U, a matroid $\mathcal{M} = (U, \mathcal{I})$, and a subset of elements $A \subseteq U$. A subset S is called feasible iff (1) S satisfies the co-matroid constraints wrt $\mathcal{M}$ (i.e. $\overline{S} \in \mathcal{I}$) and (2) S satisfies the subset constraint wrt A (i.e. $A \subseteq S$). We show the following:

▶ **Theorem 4.** *There is a 2-approximation algorithm for the problem of maximizing the* density function $d(S)$ *corresponding to a monotone supermodular function f, subject to the co-matroid and subset constraints.*

1.3 Related Work

Recently, there has been a considerable interest in the problems of optimizing submodular functions under various types of constraints. The most common constraints that are considered are *matroid constraints*, *knapsack constraints* or combinations of the two varieties. Thus for instance, Calinescu et. al [5] considered the problem of maximizing a *monotone* submodular function subject to a matroid constraint. They provide an algorithm and show that it yields a $(1 - 1/e)$-approximation: this result is essentially optimal (also see the recent paper [8] for a combinatorial algorithm for the same). Goemans and Soto [10] consider the problem of minimizing a *symmetric* submodular function subject to arbitrary *matroid* constraints. They prove the surprising result that this problem can be solved in polynomial time. In fact, their result extends to the significantly more general case of *hereditary constraints*; the problem of extending our results to *arbitrary* hereditary functions is left open.

The density functions that we consider may be considered as "close" to the notion of supermodular functions. To the best of our knowledge, the general question of *maximizing* density functions subject to a (co-)matroid constraint has never been considered before.

1.4 Proof Techniques

We employ a greedy algorithm to prove Theorems 1 and 2. Khuller and Saha [14] and Gajewar and Sarma [9] had considered a natural greedy algorithm for the problem of maximizing graph density subject to co-matroid constraints corresponding to the cardinality matroid and partition matroid respectively. Our greedy algorithm can be viewed as a natural abstraction of the greedy algorithm to the generalized scenario of arbitrary monotone supermodular functions. However, our analysis is different from that in [14, 9]. In both of the earlier papers

[14, 9], a particular stopping condition is employed to define a set D_ℓ useful in the analysis. For instance, in Section 4.1 of [9] they define D_ℓ using the optimal set H^* directly. We choose a different *stopping condition* to define the set D_ℓ; it turns out that this choice is crucial for achieving a 2-factor guarantee. Another reason for our improvement is the following: a straightforward generalization of the arguments given in [9] (to the scenario of arbitrary monotone supermodular functions) would imply a version of Claim 4 with a factor of $d^*/4$ (instead of $d^*/2$ as provided in Claim 4).

We prove Theorem 3 using LP-based techniques. Our technique also provides another proof of the basic result that graph density is computable in polynomial time. The proof method is inspired by Iwata and Nagano [13].

1.5 Organization

We present the relevant definitions in Section 2. We proceed to give the proof of Theorem 1 in Section 3. The proof of Theorem 3 is presented in Section 4. For space considerations, we include the proofs of Theorems 2 and 4 in a fuller version of the paper available at [6].

2 Preliminaries

In this paper, we will use the following notation: given *disjoint* sets A and B we will use $A+B$ to serve as shorthand for $A \cup B$. Vice versa, when we write $A+B$ it will hold implicitly that the sets A and B are disjoint.

Monotone: A set function f is called *monotone* if $f(S) \leqslant f(T)$ whenever $S \subseteq T$.

Supermodular: A set function $f : 2^U \to \mathbb{R}^+$ over a universe U is called *supermodular* if the following holds for any two sets $A, B \subseteq U$:

$$f(A) + f(B) \leqslant f(A \cup B) + f(A \cap B)$$

If the inequality holds (for every A, B) with the sign reversed, then the function f is called *submodular*. In this paper, we will use the following equivalent definition of supermodularity: given disjoint sets A, B and C,

$$f(A + C) - f(A) \leqslant f(A + B + C) - f(A + B)$$

We can think of this as follows: the *marginal utility* of the set of elements C to the set A increases as the set becomes "larger" ($A + B$ instead of A).

It is well known (see [12, 16]) that supermodular functions can be *maximized* in polynomial time (whereas submodular functions can be minimized in polynomial time). Let us record this as:

▶ **Theorem 5.** *Any supermodular function $f : 2^U \to \mathbb{R}^+$ can be maximized in polynomial time.*

We also state the following folklore corollary:

▶ **Corollary 6.** *Given any supermodular function $f : 2^U \to \mathbb{R}^+$, we can find $\max_S \frac{f(S)}{|S|}$ in polynomial time.*

A proof of this Corollary is provided in the full version [6].

Density Function: Given a function f over U, the density of a set S is defined to be $d(S) = \frac{f(S)}{|S|}$.

Matroid: A matroid is a pair $\mathcal{M} = (U, \mathcal{I})$ where $\mathcal{I} \subseteq 2^U$, and

$i \leftarrow 1$
$H_i \leftarrow \arg\max_X \frac{f(X)}{|X|}$
$D_i \leftarrow H_i$
while D_i infeasible **do**
  $H_{i+1} \leftarrow \arg\max_{X: X \cap D_i = \emptyset} \frac{f(D_i+X)-f(D_i)}{|X|}$
  $D_{i+1} \leftarrow D_i + H_{i+1}$
  $i \leftarrow i+1$
end while
$L \leftarrow i$
for $i = 1 \rightarrow L$ **do**
  Add arbitrary vertices to D_i to make it minimal feasible
  Call the result D'_i
end for
Output the subset among the D'_i's with the highest density

Figure 1 Main Algorithm

1. (Hereditary Property) $\forall B \in \mathcal{I}, A \subset B \implies A \in \mathcal{I}$.
2. (Extension Property) $\forall A, B \in \mathcal{I} : |A| < |B| \implies \exists x \in B \setminus A : A + x \in \mathcal{I}$

Matroids are generalizations of vector spaces in linear algebra and are ubiquitous in combinatorial optimization because of their connection with greedy algorithms. Typically the sets in $\mathcal{I}$ are called *independent* sets, this being an abstraction of linear independence in linear algebra. The maximal independent sets in a matroid are called the *bases* (again preserving the terminology from linear algebra). An important fact for matroids is that all bases have equal cardinality – this is an outcome of the Extension Property of matroids.

Any matroid is equipped with a *rank function* $r : 2^U \rightarrow \mathbb{R}^+$. The rank of a subset S is defined to be the size of the *largest* independent set contained in the subset S. By the Extension Property, this is well-defined. See the excellent text by Schrijver [17] for details.

Two commonly encountered matroids are the (i) *Cardinality Matroid:* Given a universe U and $r \in \mathbb{N}$, the *cardinality matroid* is the matroid $\mathcal{M} = (U, \mathcal{I})$, where a set A is *independent* (i.e. belongs to $\mathcal{I}$) iff $|A| \leqslant r$. (ii) *Partition Matroid:* Given a universe U and a partition of U as $U_1, \cdots, U_r$ and non-negative integers $r_1, \cdots, r_t$, the *partition matroid* is $\mathcal{M} = (U, \mathcal{I})$, where a set A belongs to $\mathcal{I}$ iff $|A \cap U_i| \leqslant r_i$ for all $i = 1, 2, \cdots, t$.

Convex Programs: We will need the definition of a convex program, and that they can be solved to arbitrary precision in polynomial time, via the ellipsoid method(see [12]). We refer the reader to the excellent text [4].

3 Proof of Theorem 1

We first present the algorithm and then its analysis. To get started, we describe the intuition behind the algorithm.

Note that co-matroid constraints are *upward monotone*: if a set S is feasible for such constraints, then any *superset* of S is also feasible. Thus, it makes sense to find a *maximal* subset of U with the maximum density. In the following description of the algorithm, one may note that the sets $D_1, D_2, \cdots, D_i$ are an attempt to find the maximal subset with the largest density. Given this rough outline, the algorithm is presented in Figure 1.

We note that we can find the maximum $\max_{X: X \cap D_i = \emptyset} \frac{f(D_i+X)-f(D_i)}{|X|}$ in polynomial time. This is because the function $f(D_i + X)$ for a fixed D_i is supermodular (and we appeal to

Corollary 6).

Let H^* denote the optimal solution, i.e. the subset that maximizes the density $d(S)$ subject to the co-matroid constraints. Let d^* denote the optimal density, so that $f(H^*) = d^* \cdot |H^*|$.

We can make the following easy claim:

▶ Claim 1. The subset D_1 obeys the inequality $d(D_1) \geqslant d^*$.

This is because D_1 is the densest subset in the universe U, while d^* is the density of a specific subset H^*.

In the following, we will have occasion to apply the following lemmas.

▶ **Lemma 7.** *Let $a, b, c, d, \theta \in \mathbb{R}^+$ be such that the inequalities $\frac{a}{b} \geqslant \theta$ and $\frac{c}{d} \geqslant \theta$ hold. Then it is true that $\frac{a+c}{b+d} \geqslant \theta$. Thus, if $\frac{a}{b} \geqslant \frac{c}{d}$, then $\frac{a+c}{b+d} \geqslant \frac{c}{d}$ (by setting $\theta = \frac{c}{d}$).*

Also,

▶ **Lemma 8.** *Let $a, b, c, d \in \mathbb{R}^+$ be real numbers such that $\frac{a}{b} \geqslant \frac{c}{d}$ holds.*

- *Suppose $a \geqslant c$, $b \geqslant d$. Then the inequality $\frac{a-c}{b-d} \geqslant \frac{a}{b}$ holds.*
- *Suppose $c \geqslant a$, $d \geqslant b$. Then the inequality $\frac{c}{d} \geqslant \frac{c-a}{d-b}$ holds.*

We make the following claim:

▶ Claim 2. The sequence of subsets $D_1, D_2, \cdots, D_L$ obeys the following ordering:

$$\frac{f(D_1)}{|D_1|} \geqslant \frac{f(D_2) - f(D_1)}{|D_2| - |D_1|} \geqslant \cdots \geqslant \frac{f(D_{i+1}) - f(D_i)}{|D_{i+1}| - |D_i|} \geqslant \cdots \geqslant \frac{f(D_L) - f(D_{L-1})}{|D_L| - |D_{L-1}|}$$

Proof. Consider any term in this sequence, say $\frac{f(D_{i+1}) - f(D_i)}{|D_{i+1}| - |D_i|}$. Note that H_{i+1} was chosen as arg max of $\frac{f(D_i + X) - f(D_i)}{|X|}$. Therefore, $\max_X \frac{f(D_i+X)-f(D_i)}{|X|} = \frac{f(D_{i+1})-f(D_i)}{|D_{i+1}|-|D_i|}$. Hence this quantity is larger than $\frac{f(D_{i+2})-f(D_i)}{|D_{i+2}|-|D_i|}$ (as long as D_{i+2} is well defined). Now from the second part of Lemma 8, we get that

$$\frac{f(D_{i+1}) - f(D_i)}{|D_{i+1}| - |D_i|} \geqslant \frac{f(D_{i+2}) - f(D_i)}{|D_{i+2}| - |D_i|} \geqslant \frac{f(D_{i+2}) - f(D_{i+1})}{|D_{i+2}| - |D_{i+1}|}$$

◀

Via an application of Lemma 7, we then have:

▶ Claim 3. Given any i $(1 \leqslant i \leqslant L)$, the following holds:

$$\frac{f(D_i)}{|D_i|} \geqslant \frac{f(D_i) - f(D_{i-1})}{|D_i| - |D_{i-1}|}$$

Proof. We will the prove the statement by induction.
Base Case: We implicitly assume that $D_0 = \emptyset$, and hence the case for $i = 1$ holds.
Induction Step: Assume the statement by induction for $i = k$, and we prove it for $i = k+1$. Thus, by hypothesis we have

$$\frac{f(D_k)}{|D_k|} \geqslant \frac{f(D_k) - f(D_{k-1})}{|D_k| - |D_{k-1}|}$$

Now by Claim 2 we have that

$$\frac{f(D_k) - f(D_{k-1})}{|D_k| - |D_{k-1}|} \geqslant \frac{f(D_{k+1}) - f(D_k)}{|D_{k+1}| - |D_k|}$$

Thus,

$$\frac{f(D_k)}{|D_k|} \geqslant \frac{f(D_{k+1}) - f(D_k)}{|D_{k+1}| - |D_k|}$$

Applying Lemma 7, we get:

$$\frac{f(D_{k+1})}{|D_{k+1}|} \geqslant \frac{f(D_{k+1}) - f(D_k)}{|D_{k+1}| - |D_k|}$$

Thus we have proven the Claim by induction. ◀

The analysis will be broken up into two parts. We will consider the set D_ℓ in the sequence $D_1, D_2, \cdots, D_L$ such that the following hold:

$$\frac{f(D_\ell) - f(D_{\ell-1})}{|D_\ell| - |D_{\ell-1}|} \geqslant \frac{d^*}{2}$$

but

$$\frac{f(D_{\ell+1}) - f(D_\ell)}{|D_{\ell+1}| - |D_\ell|} < \frac{d^*}{2}$$

Since $d(D_1) \geq d^*$ by Claim 1, such an ℓ will exist or $\ell = L$. If $\ell = L$, then we have a feasible solution D_L with the property that $\frac{f(D_L)-f(D_{L-1})}{|D_L|-|D_{L-1}|} \geqslant \frac{d^*}{2}$. Therefore, by Claim 3 we have that $d(D_L) \geqslant \frac{d^*}{2}$ and we are done in this case.

So we may assume that $\ell < L$ so that D_ℓ is *not* feasible. In this case, we will prove that D'_ℓ has the correct density, i.e. that $d(D'_\ell) \geqslant \frac{d^*}{2}$.

To this end, we will prove two facts about D_ℓ and that will yield the desired result:

▶ Claim 4.

$$f(D_\ell) - f(D_\ell \cap H^*) \geqslant \frac{d^*}{2}(|D_\ell| - |D_\ell \cap H^*|)$$

Proof. Note that $D_\ell = H_1 + H_2 + \cdots + H_\ell$. For brevity, for $1 \leqslant i \leqslant \ell$, denote $H_i \cap H^*$ as A_i (thus, $A_i \subseteq H_i$ for every i). Thus, $D_\ell \cap H^* = A_1 + A_2 + \cdots + A_\ell$.

We will prove the following statement by induction on i (for $1 \leqslant i \leqslant \ell$):

$$f(H_1+H_2+\cdots+H_i) - f(A_1+A_2+\cdots+A_i) \geqslant \frac{d^*}{2}(|H_1+H_2+\cdots+H_i| - |A_1+A_2+\cdots+A_i|)$$

Base Case: For $i = 1$, we have to prove that:

$$\frac{f(H_1) - f(A_1)}{|H_1| - |A_1|} \geqslant \frac{d^*}{2}$$

Since H_1 is the *densest* subset, we have

$$\frac{f(H_1)}{|H_1|} \geqslant \frac{f(A_1)}{|A_1|}$$

and we may apply (the first part of) Lemma 8 to obtain the desired.

Induction Step: Assume the statement to be true for i, and we will prove it for $i + 1$.

Consider the following chain:

$$\frac{f(H_1+\cdots+H_i+H_{i+1})-f(H_1+\cdots+H_i)}{|H_{i+1}|} \overset{H_{i+1}\arg\max}{\geqslant} \frac{f(H_1+\cdots+H_i+A_{i+1})-f(H_1+\cdots+H_i)}{|A_{i+1}|} \overset{supermodular}{\geqslant} \frac{f(A_1+\cdots+A_i+A_{i+1})-f(A_1+\cdots+A_i)}{|A_{i+1}|}$$

We would now like to apply Lemma 8 to the first and last terms in the above chain. To this end, let us check the preconditions:

$$f(H_1+\cdots+H_i+H_{i+1})-f(H_1+\cdots+H_i) \overset{monotone}{\geqslant} f(H_1+\cdots+H_i+A_{i+1})-f(H_1+\cdots+H_i) \overset{supermodular}{\geqslant} f(A_1+\cdots+A_i+A_{i+1})-f(A_1+\cdots+A_i)$$

Since, clearly $|H_{i+1}| \geqslant |A_{i+1}|$, the preconditions in Lemma 8 hold and we have:

$$\frac{f(H_1+\cdots+H_{i+1})-f(A_1+\cdots+A_{i+1})-f(H_1+\cdots+H_i)+f(A_1+\cdots+A_i)}{|H_{i+1}|-|A_{i+1}|} \geqslant \frac{f(H_1+\cdots+H_i+H_{i+1})-f(H_1+\cdots+H_i)}{|H_{i+1}|} \geqslant \frac{d^*}{2}$$

Applying Lemma 7 to the first term in the above chain and the induction statement for i, we obtain the desired result for $i+1$. Hence done. ◀

The next claim lower bounds the value $f(D_\ell \cap H^*)$.

Building up to the Claim, let us note that $D_\ell \cap H^* \neq \emptyset$. If the intersection were empty, then H^* is a subgraph of density d^*, and so $H_{\ell+1}$ would be a subgraph of density at least d^*. But then,

$$\frac{f(D_\ell+H_{\ell+1})-f(D_\ell)}{|H_{\ell+1}|} \overset{supermodular}{\geqslant} \frac{f(H_{\ell+1})}{|H_{\ell+1}|} \geqslant d^*$$

But this contradicts the choice of D_ℓ.

▶ Claim 5.

$$f(D_\ell \cap H^*) \geqslant \frac{d^*}{2}|D_\ell \cap H^*| + \frac{d^*}{2}|H^*|$$

Proof. Let $X = H^* - D_\ell \cap H^*$. Then, $X \cap D_\ell = \emptyset$ and $D_\ell + X = D_\ell \cup H^*$. Then by definition of D_ℓ, we know that $\frac{f(D_\ell+X)-f(D_\ell)}{|X|} \leqslant \frac{f(D_{\ell+1})-f(D_\ell)}{|D_{\ell+1}|-|D_\ell|} < d^*/2$. Thus, $f(D_\ell \cup H^*) - f(D_\ell) \leqslant \frac{d^*}{2}(|H^*| - |D_\ell \cap H^*|)$.

Therefore, $f(D_\ell \cup H^*) + f(D_\ell \cap H^*) \leqslant f(D_\ell) + f(D_\ell \cap H^*) + \frac{d^*}{2}(|H^*| - |D_\ell \cap H^*|)$.

Applying supermodularity we have that $f(D_\ell \cup H^*) + f(D_\ell \cap H^*) \geqslant f(D_\ell) + f(H^*)$. Thus, cancelling $f(D_\ell)$ gives us that $f(D_\ell \cap H^*) + \frac{d^*}{2}(|H^*| - |D_\ell \cap H^*|) \geqslant f(H^*)$. The claim follows by observing that $d^* = \frac{f(H^*)}{|H^*|}$. ◀

Note that this claim also implies that the density of the set $D_\ell \cap H^*$ is at least d^*. Intuitively, $D_\ell \cap H^*$ is a subset that has "enough f-value" as well as a "good" density.

We may now combine the statements of Claim 4 and Claim 5 to get the following chain of inequalities:

$$f(D_\ell) \overset{Claim\ 4}{\geqslant} f(D_\ell \cap H^*) + \frac{d^*}{2}|D_\ell| - \frac{d^*}{2}|D_\ell \cap H^*| \overset{Claim\ 5}{\geqslant} \frac{d^*}{2}|D_\ell| + \frac{d^*}{2}|H^*|$$

Consider D'_ℓ: this is obtained from D_ℓ by adding suitably many elements to make D_ℓ feasible. Let r be the minimum *number* of elements to be added to D_ℓ so as to make it feasible. Since H^* is a feasible solution too, clearly, $r \leqslant |H^*|$. With this motivation, we define the *Extension Problem* for a matroid $\mathcal{M}$. The input is a matroid $\mathcal{M} = (U, \mathcal{I})$ and a subset $A \subseteq U$. The goal is to find a subset T of minimum cardinality such that $A \cup T \in \mathcal{I}$. Lemma 9 shows that we can find such a subset T in polynomial time. Thus, we would have that:

$$d(D'_\ell) = \frac{f(D'_\ell)}{|D_\ell| + r} \geqslant \frac{f(D_\ell)}{|D_\ell| + r} \geqslant \frac{f(D_\ell)}{|D_\ell| + |H^*|} \geqslant d^*/2$$

and we are done with the proof of Theorem 1, modulo the proof of Lemma 9.

We proceed to present the lemma and its proof:

▶ **Lemma 9.** *The Extension Problem for matroid $\mathcal{M}$ and subset A can be solved in polynomial time.*

Proof. The proof considers the *base polyhedron* of the matroid (see the text by Schrijver [17]). We will have a variable x_i for each element $i \in U \setminus A$, where $x_i = 1$ would indicate that we pick the element i in our solution T. For brevity, we will also maintain a variable y_i that indicates whether i is *absent* from the solution T. Thus for every i, we will maintain that $x_i + y_i = 1$. Given an arbitrary set S, we will let $r(S)$ denote the *rank* of the subset S in the matroid $\mathcal{M}$.

The following is a valid integer program for the Extension Problem (where $y(S)$ is shorthand for $\sum_{i \in S} y_i$). The linear program to the right is the relaxation of the integer program, and with variables x_i eliminated.

$$\text{IP}_1: \quad \begin{array}{lll} \min & \displaystyle\sum_{i \in U} x_i & \\ \text{s.t.} & x_i + y_i = 1 & \text{for all } i \in U \\ & y(S) \leqslant r(S) & \text{for all } S \subseteq U \\ & x_i = 1 & \text{for all } i \in A \\ & x_i, y_i \in \{0,1\} & \text{for all } i \in U\ . \end{array} \qquad \text{LP}_1: \quad \begin{array}{lll} \min & \displaystyle\sum_{i \in U} (1 - y_i) & \\ \text{s.t.} & y(S) \leqslant r(S) & \text{for all } S \subseteq U \\ & y_i = 0 & \text{for all } i \in A \\ & y_i \geqslant 0 & \text{for all } i \in U\ . \end{array}$$

The linear program LP_1 can also be formulated as a maximization question. To be precise, let $\text{VAL}(\text{LP}_1)$ denote the *value* of the program LP_1. Then $\text{VAL}(\text{LP}_1) = |U| - \text{VAL}(\text{LP}_2)$, where LP_2 is as follows:

$$\text{LP}_2: \quad \begin{array}{lll} \max & \displaystyle\sum_{i \in U} y_i & \\ \text{s.t.} & y(S) \leqslant r(S) & \text{for all } S \subseteq U \\ & y_i = 0 & \text{for all } i \in A \\ & y_i \geqslant 0 & \text{for all } i \in U\ . \end{array}$$

Now, by folklore results in matroid theory (cf. [17]), we have that solutions to LP_2 are integral and can be found by a greedy algorithm. Thus, we can solve IP_1 in polynomial time, and this proves the statement of the Lemma. ◀

4 Proof of Theorem 3

We will present the proof for the case of the graph density function, i.e. where $f(S) = |E(S)|$. The proof for arbitrary f will require a passage to the Lovász Extension $\mathcal{L}_f(x)$ of a set function $f(S)$ and is deferred to the full version [6].

We will *augment* the LP that Charikar [7] uses to prove that graph density is computable in polynomial time. Given a graph $G = (V, E)$, there are edge variables y_e and vertex variables x_i in the LP. We are also given an auxiliary *dependency* digraph $D = (V, \vec{A})$ on the vertex set V. In the augmented LP, we also have constraints $x_i \leqslant x_j$ if there is an arc from i to j in the digraph $D = (V, \vec{A})$. The DENdep problem is modelled by the linear program LP_3.

$$\text{LP}_3: \quad \begin{aligned} \max \quad & \sum_{e \in E} y_e \\ \text{s.t.} \quad & \sum_i x_i = 1 \\ & y_e \leqslant x_i && \forall e \sim i, e \in E \\ & x_i \leqslant x_j && \forall (i,j) \in \vec{A} \\ & x_i \geqslant 0 && \forall i \in V(G) \end{aligned} \qquad \text{CP}_1: \quad \begin{aligned} \max \quad & \sum_{e=(i,j) \in E} \min\{x_i, x_j\} \\ \text{s.t.} \quad & \sum_i x_i = 1 \\ & x_i \leqslant x_j && \forall (i,j) \in \vec{A} \\ & x_i \geqslant 0 && \forall i \in V(G) \end{aligned}$$

Suppose we are given an optimal solution H^* to the DENdep problem. Let $\text{VAL}(\text{LP}_3)$ denote the feasible value of this LP: we will prove that $\text{VAL}(\text{LP}_3) = d(H^*)$.

$\text{VAL}(\text{LP}_3) \geqslant d(H^*)$:

We let $|H^*| = \ell$, and $x_i = 1/\ell$ for $i \in H^*$, and 0 otherwise. Likewise, we set $y_e = 1/\ell$ for $e \in E(H^*)$, and 0 otherwise. Note that H^* is feasible, so if $a \in H^*$ and $(a, b) \in \vec{A}$, then it also holds that $b \in H^*$. We may check that the assignment x and y is feasible for the LP. So, $d(H^*) = \frac{|E(H^*)|}{\ell}$ is achieved as the value of a feasible assignment to the LP.

$\text{VAL}(\text{LP}_3) \leqslant d(H^*)$:

In the rest of the proof, we will prove that there exists a subgraph H such that $\text{VAL} \leqslant d(H)$. First, it is easy to observe that in any optimal solution of the above LP, the variables y_e will take the values $\min\{x_i, x_j\}$ where $e = (i, j)$. Thus, we may eliminate the variables y_e from the program LP_3 to obtain the program CP_1. We claim that CP_1 is a convex program. Given two concave functions, the min operator preserves concavity. Thus, the objective function of the above modified program is concave. Hence we have a convex program: here, the objective to be *maximized* is concave, subject to linear constraints. We may solve the program CP_1 and get an output optimal solution x^*. Relabel the vertices of V such that the following holds: $x^*_1 \geqslant x^*_2 \geqslant \cdots \geqslant x^*_n$. If there are two vertices with (modified) indices a and b where $a < b$ and there is an arc $(a, b) \in \vec{A}$, then we have the equalities $x^*_a = x^*_{a+1} = \cdots = x^*_b$. We will replace the inequalities in the program CP_1 as follows:

$$\text{LP}_4: \quad \begin{aligned} \max \quad & \sum_{e=(i,j) \in E : i<j} x_j \\ \text{s.t.} \quad & \sum_i x_i = 1 \\ & x_i \geqslant x_{i+1} && \text{for all } i \in \{1, 2, \cdots, (n-1)\} \\ & x_n \geqslant 0 && . \end{aligned}$$

where some of the inequalities $x_i \geqslant x_{i+1}$ may be equalities if there is an index a with $a \leqslant i$ and an index b with $b \geqslant (i+1)$ such that $(a, b) \in \vec{A}$. Note also that because of the ordering of the variables of this LP, the objective function also simplifies and becomes a linear function. Clearly x^* is a feasible solution to this LP. Thus the value of this LP is no less than the

value of CP_1. Consider a BFS x to LP_4. The program LP_4 has $(n+1)$ constraints, and n variables. Given the BFS x, call a constraint *non-tight* if it does not hold with equality under the solution x. Thus, there may be at most one *non-tight* constraint in LP_4. In other words, there is at most one constraint $x_i \geqslant x_{i+1}$ that is a strict inequality. This, in turn, implies that all the non-zero values in x are equal. Let there be ℓ such non-zero values. From the equality $\sum_i x_i = 1$, we get that each non-zero $x_i = 1/\ell$. Let H denote the set of indices i that have $x_i > 0$. The objective value corresponding to this BFS x is $|E(H)|/\ell = d(H)$.

Thus we have proven that $d(H) \geqslant \mathrm{VAL}(LP_4) \geqslant \mathrm{VAL}(CP_1) = \mathrm{VAL}(LP_3)$, as required. This completes the proof of Theorem 3.

Remarks about the proof:

- We remark that the objective in the convex program CP_1 is precisely the Lovász Extension $\mathcal{L}_f(x)$ for the specific function $f = |E(S)|$. Thus our proof shows that the LP provided by Charikar [7] is precisely the Lovász Extension for the supermodular function $|E(S)|$.
- Note that there are other proofs possible for this result. For instance, one can follow the basic argument of Charikar to show that LP_3 satisfies $d(H^*) = \mathrm{VAL}(LP_3)$. The proof we provide above is new, and is inspired by the work of Iwata and Nagano [13].
- Via our proof, we also prove that any BFS for the basic graph density LP has the property that all the non-zero values are equal. This fact is not new: it was proven by Khuller and Saha [14] but we believe our proof of this fact is more transparent.

5 Acknowledgements

We gratefully acknowledge helpful discussions with Aman Dhesi, Raghav Kulkarni and Amit Kumar about the topic.

References

1 N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, New York, 1992.
2 R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, pages 25–37, 2009.
3 A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $o(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010.
4 S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
5 G. Călinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.
6 V. T. Chakaravarthy, N. Modani, S. R. Natarajan, S. Roy, and Y. Sabharwal. Density functions subject to a co-matroid constraint. *CoRR*, abs/1207.5215, 2012.
7 M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 84–95, 2000.
8 Y. Filmus and J. Ward. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *FOCS*, 2012.
9 A. Gajewar and A. Das Sarma. Multi-skill collaborative teams based on densest subgraphs. In *SDM*, pages 165–176, 2012.
10 M. X. Goemans and J. A. Soto. Symmetric submodular function minimization under hereditary family constraints. *CoRR*, abs/1007.2140, 2010.
11 A. V. Goldberg. Finding a maximum density subgraph. Technical report, UC Berkeley, 1984.
12 M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

13 S. Iwata and K. Nagano. Submodular function minimization under covering constraints. In *FOCS*, pages 671–680, 2009.

14 S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP*, 2009.

15 B. Saha, A. Hoch, S. Khuller, L. Raschid, and X-N. Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *RECOMB*, pages 456–472, 2010.

16 A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Comb. Theory, Ser. B*, 80(2):346–355, 2000.

17 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

18 D.M. Topkis. *Supermodularity and Complementarity*. Princeton University Press, 1998.

Efficient on-line algorithm for maintaining k-cover of sparse bit-strings

Amit Kumar[1], Preeti R. Panda[1], and Smruti Sarangi[1]

1 **Indian Institute of Technology, Delhi**
New Delhi, India

Abstract

We consider the on-line problem of representing a sparse bit string by a set of k intervals, where k is much smaller than the length of the string. The goal is to minimize the total length of these intervals under the condition that each 1-bit must be in one of these intervals. We give an efficient greedy algorithm which takes time $O(\log k)$ per update (an update involves converting a 0-bit to a 1-bit), which is independent of the size of the entire string. We prove that this greedy algorithm is 2-competitive. We use a natural linear programming relaxation for this problem, and analyze the algorithm by finding a dual feasible solution whose value matches the cost of the greedy algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases On-line algorithms, string algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.249

1 Introduction

In many applications, we need to maintain representations of sparse bit vectors. Such representations should be compact, and one should be able to update them efficiently even in the on-line setting. One way of achieving this is to cluster the 1-bits into small number of intervals. More formally, given a long bit vector and a (relatively) small integer k, we would like to find a set of k (disjoint) intervals such that each 1-bit is in one of these intervals. Such a representation can lead to significant compression of data, and fast retrieval of information. The goal here would be find these intervals such that they contain as few 0-bits as possible, i.e., we would like to minimize the total length of these intervals. We call this the k-`Cover` problem. Such representations have applications in many areas of computer science including compilers, architecture and databases. In this paper, we address this problem in the online or streaming setting: in each time step, one of the 0-bits becomes 1, and we would like to update our representation in time depending on k only. Indeed, in many applications n could be very large, and even running time of $O(\log n)$ per time step could be prohibitive.

Toussi et al. [1] proposed a new algorithm for compressing the information stored in a traditional *points-to*[2] analysis pass in a compiler. In a *points-to* analysis, a compiler first makes a list of all possible objects that are in the scope of a function, and then marks a subset of them that can be possibly accessed by the function under consideration. There are both off-line and online variants of this algorithm. If the analysis is done in one pass, then we can consider it as an on-line version. If the analysis takes multiple passes especially after taking

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 249–256

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

inter-procedural information into account, we can consider it to be an off-line variant. Toussi et al. [1] proposed to save this information in the form of a bit vector. Each bit corresponds to a unique object/variable in the system. If a bit is 1, then there is a possibility that a given function might access a certain variable. This bit vector can further be compressed to form a *ranged bit vector* that saves only intervals containing the 1-bits. The authors do entertain the possibility of having intermittent 0's similar to our definition of a k-`Cover` problem. In this case, if we decrease the value of k, then we have better compression, yet the compiler is more conservative.

In the area of scientific databases, Sinha et al. [7, 8] reported the use of a scheme similar to the k-`Cover` problem. They consider a variant of binning for bit vector databases that is tailored for storing sparse vectors. They consider a hierarchical structure, where each level stores a bit vector as a sequence of intervals. At the highest level, they store a set of k intervals, where k might possibly be a variable. To filter out false positives, they introduce additional levels to store each interval further as a set of intervals. The last level does not have any false positives. This helps them answer range queries with different degrees of approximation. There is clearly a tradeoff between time and accuracy.

In the area of computer architecture, similar ideas have been used to compress post-silicon debug information [9, 10, 11]. In each epoch, the authors propose to mark a cache line with 1 if it is written to in that interval. The default state of each line is 0. Vishnoi et al. [11] proposed to save this information in the form of a large bit vector. Panda et al. [10] tried to further improve upon this work by trying to compress the bit vector using LZW compression. We can also use the idea of covering by k intervals in this context, because the bit vectors were found to be fairly sparse.

It is easy to solve the off-line version of the k-`Cover` problem. Indeed, the following simple greedy algorithm works – define a gap in a bit-string as a maximal sequence of 0's. Then, the optimal solution should construct the k intervals such that it leaves out the largest $k-1$-gaps (excluding the leftmost and rightmost gaps). In fact, we can easily convert this to an on-line algorithm which maintains a suitable data-structure for storing the gaps – each update can be achieved in $O(\log n)$-time. But as pointed out above, in many applications, k could be much smaller than n, and we would like an algorithm whose running time per update depends on k only.

Our Results We give a simple greedy 2-competitive algorithm for the k-`Cover` problem. Further our algorithm only takes $O(\log k)$-time per update. The analysis of our algorithm requires non-trivial ideas. We do not have any natural charging or potential function based arguments for bounding the competitive ratio of our algorithm. Instead, we use a dual fitting argument. We first write a natural LP relaxation for this problem (which has integrality gap of 1). We then show that we can assign values to the variables in the dual LP such that the values are feasible and the objective function is close to the cost of the greedy algorithm. We expect our techniques to be useful for analyzing other on-line string algorithms.

Related Work There is a vast amount of literature on string algorithms and string compression techniques [5, 6]. To the best of our knowledge, this problem has not been considered before in this context. Our problem is a special case of the clustering problem which minimizes the sum of cluster diameters [4]. A constant factor approximation algorithm is known for the latter problem (for any arbitrary metric) [3].

In Section 2, we define the problem formally and describe the greedy algorithm. Subsequently, we analyze the greedy algorithm by first giving an LP relaxation for our problem

(Section 3) and then setting the dual variables (Section 4). Finally, we show that the greedy algorithm cannot be better than 2-competitive (Section 5).

2 Preliminaries

We are given a sequence of n bits. At time t, let S_t denote the sequence of these bits. Let $b_i^{(t)}$ denote the i^{th} bit at time t. Initially, in S_0, all the bits are 0. At each point of time t, one of the bits which was 0 at time $(t-1)$ becomes 1. The input instance also specifies a parameter k. The algorithm needs to maintain a set of k disjoint intervals (which we will call *covers*) at all points of time t – these covers must satisfy the property that any bit which is 1 should lie in one of these covers. Let $C_1^{(t)}, \ldots, C_k^{(t)}$ denote the set of covers at time t. For an interval I, $l(I)$ and $r(I)$ will denote the left and the right end-points of I. Define a *gap* as the sequence of 0's between two consecutive covers. We shall use $G_i^{(t)}$ to denote the gap between $C_i^{(t)}$ and $C_{i+1}^{(t)}$, i.e., the open interval $(r(C_i^{(t)}), l(C_{i+1}^{(t)}))$. Our goal is to minimize the total length of covers at any time, i.e., $\sum_{i=1}^{r} |C_i^{(t)}|$. Let $\texttt{opt}_t$ be the value of the optimal solution at time t.

We propose a simple greedy algorithm $\mathcal{A}$. Suppose at time t, the bit d_t becomes one. If d_t lies in one of the covers at time t, we do not need to do anything. Otherwise, we tentatively create one more cover consisting of the singleton element d_t. Now we have a set of $k+1$ covers (which cover all the 1 bits). Among these $k+1$ covers, we find two consecutive covers with the minimum gap between them – call these C and C', and assume that C lies to the left of C'. We merge them into a single cover, i.e., we replace C and C' by $[l(C), r(C')]$. Thus we again have only k covers at time $t+1$ and they contain all the 1 bits. Note that we always grow the current covers.

For the sake of simplifying the analysis, we shall make the following assumption : at any point of time t, the bit d_t that becomes 1 does not lie in any of the existing covers. This assumption is without loss of generality – if we forbid such bits from becoming 1, we do not affect our algorithm $\mathcal{A}$, and this can only reduce the value of the optimum solution.

3 A Linear Programming Relaxation and its dual

In order to bound the cost of $\mathcal{A}$, we need to have a handle on the value of the optimum (for any time t). Although it is easy to understand what the optimum is, it will be more convenient to deal with a lower bound given by a linear programming relaxation. For a time t, consider the following LP relaxation.

$$\min \sum_I l(I) x_I^{(t)} \tag{1}$$

$$\sum_{I: i \in I} x_I^{(t)} \geq 1 \qquad \text{for all bits } i \text{ such that } b_i^{(t)} = 1 \tag{2}$$

$$\sum_I x_I^{(t)} \leq k \tag{3}$$

$$x_I^{(t)} \geq 0 \qquad \text{for all intervals } I$$

For any interval I, it has a variable $x_I^{(t)}$, which is supposed to be 1 if I gets chosen in the

set of covers at time t, otherwise it should be 0. Let $l(I)$ denote the length of I. The objective function in (1) measures the total length of the chosen intervals. The first constraint (2) says that for any bit which is 1, one of the covers must contain it. The second constraint (3) requires that we can choose only k intervals. Given any solution to our problem instance, we can define a solution to this LP (where we set $x_I^{(t)}$ to 1 iff we take I in our set of covers at time t) which satisfies all the constraints. Therefore, the optimal value of this LP is a lower bound on $\texttt{opt}_t$.

Now, we write the dual of the above LP. Note that the optimal value of the dual is at most the optimal value of the LP above. We have variables $\alpha_i^{(t)}$ for all bits i such that $b_i^{(t)} = 1$ (corresponding to constraints (2)) and $\beta^{(t)}$ for constraint (3).

$$\max \sum_{i:b_i^{(t)}=1} \alpha_i^{(t)} - k\beta^{(t)} \tag{4}$$

$$\sum_{i:i\in I, b_i^{(t)}=1} \alpha_i^{(t)} - \beta^{(t)} \leq l(I) \quad \text{for all intervals } I \tag{5}$$

$$\alpha_i^{(t)}, \beta^{(t)} \geq 0$$

Our goal is to show that for any t, we can construct a feasible solution $\alpha_i^{(t)}, \beta^{(t)}$ to the dual LP above such that the cost of the solution is at least half of the cost of our algorithm $\mathcal{A}$ at time t. Once we can show this, then the cost of our algorithm will be at most twice the cost of the dual LP, and hence at most twice the cost of the primal LP. This will show that $\mathcal{A}$ is 2-competitive. In the next section, we show how to set these dual variables.

4 Setting the Dual Variables

Recall that $C_1^{(t)}, \ldots, C_k^{(t)}$ denote the set of covers constructed by $\mathcal{A}$ at time t. We shall assume that $t \geq k$, otherwise the greedy algorithm will also output the optimal covers consisting of singleton elements. Further $G_i^{(t)}$ denotes the gap between $C_i^{(t)}$ and $C_{i+1}^{(t)}$. Note that there will be $k-1$ gaps. At time t, let $\gamma^{(t)}$ denote the length of the smallest gap. We set $\beta^{(t)} = \max_{k\leq t'\leq t} \gamma^{(t)}$. The values $\alpha_i^{(t)}$ will be more subtle. Instead of specifying their exact value, we will write some invariants which will be satisfied at all times. Once we prove these invariants, it will be easy to check that the dual constraints are satisfied. Suppose bit $b_i^{(t)} = 1$ and it lies in cover $C_r^{(t)}$. The values $\alpha_i^{(t)}$ will satisfy the following conditions :

(i) If $b_i^{(t)}$ is the right end-point of $C_r^{(t)}$, then $\alpha_i^{(t)} = \min(\beta^{(t)}, |G_i^{(t)}|) + 1$ (if $b_i^{(t)}$ lies in the rightmost cover, we treat $|G_i^{(t)}|$ as infinity).

(ii) If $b_i^{(t)}$ is not the right end-point of $C_r^{(t)}$, let $b_u^{(t)}$ be the next bit to the right of $b_i^{(t)}$ which is 1 (so $b_u^{(t)}$ also lies in $C_r^{(t)}$). Let $d(i,u) = u - i - 1$ be the gap between the two bits (it counts the number of 0's in between). Then $\alpha_i^{(t)}$ will have a value which will be at most $\min(\beta^{(t)} + 1, 2d(i,u) + 2)$.

(iii) $\sum_r \alpha_r^{(t)} - k\beta^{(t)}$ is at least $\sum_{r=1}^{k} |C_r^{(t)}|$.

We now prove that it is possible to set the values $\alpha_i^{(t)}$ such that the above three conditions are satisfied. We proceed by induction on t.

Base Case : Consider time $t = k$: we have exactly k bits which are 1. The k covers will contain these singleton elements each. We set $\alpha_i^{(k)} = \gamma^{(k)} + 1$ for each $i = 1, \ldots, k$. Also,

$\beta^{(k)} = \gamma^{(k)}$. Clearly, condition (i) holds. The requirements of condition (ii) do not hold for any cover, so it is vacuously true. For condition (iii), note that $\sum_r \alpha_r^{(k)} = k\gamma^{(k)} + k$, and so it is satisfied as well.

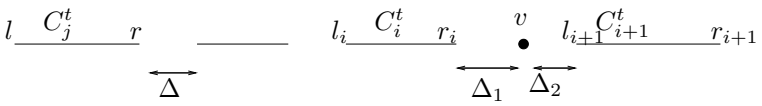


Figure 1 Covers at time t and arrival of new 1-bit at v

Induction Step : Suppose the induction hypothesis holds at some time t, i.e., there are variables $\alpha_i^{(t)}$ satisfying the three invariants. We will prove that the hypothesis holds at time $t+1$ as well. Suppose at time $t+1$, the bit b_v becomes 1. Assume that b_v lies between the covers $C_i^{(t)}$ and $C_{i+1}^{(t)}$. For the ease of notation, let us denote the end-points of these covers as $C_i^{(t)} = [l_i, r_i], C_i^{(t+1)} = [l_{i+1}, r_{i+1}]$. Let the sequence of 0's between r_i and v be of length Δ_1, and that between v and l_{i+1} be of length Δ_2 (see figure above). There are three possibilities on what our algorithm can do. We show how to set the dual variables in each of these cases.

(a) We merge v with the right end point r_i of $C_i^{(t)}$: first of all, observe that $\gamma^{(t+1)} \leq \gamma^{(t)}$, because we are only reducing the size of one of the gaps. Hence, $\beta^{(t+1)}$ stays at $\beta^{(t)}$. Also, it must be the case that $\Delta_1 \leq \Delta_2$, and $\Delta_1 \leq \gamma^{(t)}$ (because our greedy algorithm always includes the smallest gap in a cover). We change the dual variables as follows :

$$\alpha_v^{(t+1)} = 1+\min(\beta^{(t+1)}, \Delta_2), \alpha_{r_i}^{(t+1)} = 1+\Delta_1+\min(\Delta_1+\Delta_2+1, \beta^{(t+1)})-\min(\Delta_2, \beta^{(t+1)}).$$

Rest of the dual variables do not change. Let us now check the invariants. We need to check condition (i) for v because it is the new right end-point of its cover. But the definition of $\alpha_v^{(t+1)}$ is in accordance with this condition. Condition (ii) now needs to be checked for r_i : recall that $\Delta_1 \leq \gamma^t$, which is at most $\beta^{(t+1)}$. So,

$$\alpha_{r_i}^{(t+1)} \leq 1 + \Delta_1 + \Delta_1 + 1 + \min(\Delta_2, \beta^{(t+1)}) - \min(\Delta_2, \beta^{(t+1)}) = 2\Delta_1 + 2,$$

and

$$\alpha_{r_i}^{(t+1)} \leq 1 + \Delta_1 + \beta^{(t+1)} - \min(\Delta_2, \beta^{(t+1)}) \leq 1 + \beta^{(t+1)}.$$

Let us now check condition (iii). The induction hypothesis applied to (i) says that the dual value $\alpha_{r_i}^{(t)}$ was $1 + \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)})$. So,

$$\begin{aligned}\left(\sum_r \alpha_r^{(t+1)} - k\beta^{(t+1)}\right) - \left(\sum_r \alpha_r^{(t)} - k\beta^{(t)}\right) &= \alpha_v^{(t+1)} + \alpha_{r_i}^{(t+1)} - \alpha_{r_i}^{(t)} \\ &= \Delta_1 + 1,\end{aligned}$$

which is exactly the increase in the cost of our solution.

(b) We merge v with the left end point l_{i+1} of $C_{i+1}^{(t)}$: Similar to the argument above, $\Delta_2 \leq \Delta_1, \Delta_2 \leq \gamma^{(t)}$ and $\beta^{(t+1)} = \beta^{(t)}$. We again change the dual variables for v and r_i only :

$$\alpha_v^{(t+1)} = 1+\Delta_2+\min(\Delta_1+\Delta_2+1, \beta^{(t+1)})-\min(\Delta_1, \beta^{(t+1)}), \alpha_{r_i}^{(t+1)} = 1+\min(\Delta_1, \beta^{(t+1)}).$$

Let us check the invariants. Condition (i) holds for r_i again. We need to check (ii) for v. Again, note that

$$\alpha_v^{(t+1)} \leq 1 + \Delta_2 + \min(\Delta_1, \beta^{(t+1)}) + 1 + \Delta_2 - \min(\Delta_1, \beta^{(t+1)}) = 2\Delta_2 + 2,$$

and

$$\alpha_v^{(t+1)} \leq 1 + \Delta_2 + \beta^{(t+1)} - \min(\Delta_1, \beta^{k+1}) \leq 1 + \beta^{(t+1)}.$$

Finally, we see the change in the objective function. As in the previous case, $\alpha_{r_i}^{(t)} = 1 + \min(\Delta_1 + \Delta_2 + 1, \beta^t)$. So the increase in the objecive function is

$$\alpha_v^{(t+1)} + \alpha_{r_i}^{(t+1)} - \alpha_t = 1 + \Delta_2,$$

which is also the increase in our cost.

(c) We merge covers $C_j^{(t)}$ and $C_{j+1}^{(t)}$, where $j \neq i$. So, we add a new cover consisting of the singleton element v. Let Δ denote the length of the gap $G_j^{(t)}$ (between $C_j^{(t)}$ and $C_{j+1}^{(t)}$). So, $\Delta \leq \min(\Delta_1, \Delta_2)$. Further $\Delta = \gamma^t$. Also, it is possible that $\gamma^{(t+1)}$ is larger than $\gamma^{(t)}$. So $\beta^{(t+1)} = \max(\beta^{(t)}, \gamma^{(t+1)})$. Let the end-points of $C_j^{(t)}$ be $[l, r]$. For this, we consider two sub-cases.

- $\beta^{(t+1)} = \beta^{(t)}$: We set

$$\alpha_{r_i}^{(t+1)} = \min(\Delta_1, \beta^{(t+1)}) + 1, \alpha_v^{(t+1)} = \min(\Delta_2, \beta^{(t+1)}) + 1,$$

and

$$\alpha_r^{(t+1)} = 2\Delta + 1 + \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) - \min(\Delta_1, \beta^{(t+1)}) - \min(\Delta_2, \beta^{(t+1)}).$$

All other dual variables remain unchanged. Note that the bit at position r is no longer the end-point of a cover, so we need to check condition (i) for r_i and v only. Again, by definition, the conditions hold here. We need to check condition (ii) for bit r now. Observe that

$$\alpha_r^{(t+1)} \leq 2\Delta + 1 + 1 = 2\Delta + 2,$$

and because $\Delta \leq \Delta_1, \Delta_2, \beta^{(t+1)}$,

$$\begin{aligned}\alpha_r^{(t+1)} \leq 1 + \min(\Delta_1, \beta^{(t+1)}) + \min(\Delta_2, \beta^{(t+1)}) + \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) \\ - \min(\Delta_1, \beta^{(t+1)}) - \min(\Delta_2, \beta^{(t+1)}) \leq 1 + \beta^{(t+1)}.\end{aligned}$$

Finally, we calculate the change in the objective function. By induction hypothesis and condition (i), we know that

$$\alpha_r^{(t)} = \Delta + 1, \alpha_{r_i}^{(t)} = \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) + 1.$$

So, the change in objective function is

$$\begin{aligned}\alpha_r^{(t+1)} + \alpha_{r_i}^{(t+1)} + \alpha_v^{(t+1)} - \alpha_r^{(t)} - \alpha_{r_i}^{(t)} &= 2\Delta + 3 + \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) - \Delta \\ &\quad -1 - \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) - 1 \\ &= \Delta + 1,\end{aligned}$$

which is exactly the increase in the cost of the algorithm.

- $\beta^{(t+1)} = \gamma^{(t+1)} > \beta^{(t)}$: It must be the case that $\Delta_1, \Delta_2 > \beta^{(t)}$. For all the right end points r_j of the covers in the solution at time $(t+1)$ (which includes the singleton element v), we set $\alpha_{r_j}^{(t+1)} = \beta^{(t+1)} + 1$. Rest of the dual variables remain unchanged. Note that the bit at position r is no longer the end-point of a cover, and so, $\alpha_r^{(t+1)} = \alpha_r^{(t)} = \Delta + 1$. Condition (i) is satisfied because gap between any two covers is at least $\beta^{(t+1)}$. Condition (ii) is also trivially satisfied – for bit r, note that the gap to the next 1 bit on the right is Δ. So, we just need to check the change in the objective function. Note that we have change the dual value of the right end point of all covers except r at time t. Also, we added a new cover $\{v\}$. Since $\alpha_i^{(t)} \leq \beta^{(t)} + 1$ for all bits

$b_i^{(t)}$ (invariant (i)), the change is

$$\begin{aligned}&\left(\sum_r \alpha_r^{(t+1)} - k\beta^{(t+1)}\right) - \left(\sum_r \alpha_r^t - k\beta^t\right)\\ &\geq k(\beta^{(t+1)}+1) - \Big((k-1)(\beta^{(t)}+1)) - k(\beta^{(t+1)} - \beta^{(t)})\Big)\\ &= \beta^{(t)} + 1 \;\geq\; \Delta + 1\end{aligned}$$

which is again the increase in our cost.

Thus, we have shown that the invariants hold at all times. We now need to show that the dual variables are approximately feasible to the dual LP.

▶ **Lemma 1.** *For any interval I and time t,*

$$\sum_{i:i\in I, b_i^{(t)}=1} \alpha_i^{(t)} - \beta^{(t)} \leq 2l(I).$$

Proof. Consider an interval $I = [a, b]$. Suppose $a \in C_j^{(t)}$ and $b \in C_l^{(t)}$ $(j \leq l)$. If $i \in I$ and $b_i = 1$, let $n(i)$ be the next bit (on the right) which is 1. Let A be the set of all bit positions which are 1 in I except for the last such bit. So if $i \in A$, then $n(i) \in I$. Also, $l(I) - 1 \geq \sum_{i\in A}(n(i) - i)$ (we are not counting the last 1-bit, so we subtract 1 from LHS). For any $i \in A$, $\alpha_i^{(t)} \leq 2(n(i) - i)$. Indeed, for any bit $i \in A$ which is not the last bit in the cover containing it, this follows from condition (ii). For bits i which are the right end-point of a cover, this follows from condition (i) (in fact, we do not need the factor 2 here). Let u be the last bit in I which is 1. By condition (ii) or (i), $\alpha_u^{(t)} \leq \beta^{(t)} + 1$. Putting, everything together, we have

$$\sum_{i:i\in I, b_i^{(t)}=1} \alpha_i^{(t)} = \sum_{i\in A} \alpha_i^{(t)} + \alpha_u^{(t)} \leq \sum_{i\in A}(n(i) - i) + \beta^{(t)} + 1 \leq 2l(I) + \beta^{(t)}.$$

This proves the lemma. ◀

The lemma shows that $\alpha_i^{(t)}/2, \beta^{(t)}/2$ are dual feasible. Condition (iii) implies that the cost of this solution is at least half the cost of our algorithm. So our algorithm is 2-competitive.

5 Lower Bound for the Greedy Algorithm

In this section, we show that the competitive ratio of our algorithm can be close to 2.

▶ **Lemma 2.** *Given any constant $c < 2$, there is an instance for which our algorithm has competitive ratio at least c.*

Proof. We consider a bit string of length n, where n is large enough. We also fix a parameter k (which is much smaller than n). Our instance is required to maintain k covers. Initially all bits are 0. Now suppose the first $k + 1$ rounds, the bits which become 1 are at positions $0, D, 2D, \ldots, kD$ for some parameter D (assume $n > kD$). Our greedy algorithm must have some interval which contains two of these 1's. Assume that one of the greedy covers includes $[iD, (i + 1)D]$ for some i. Now, we pick a j different from i, and in the next $D - 1$ rounds,

we set all the bits in the positions $[jD, (j+1)D]$ to 1. So the greedy algorithm must have cost at least $2D$. However, the optimal off-line algorithm will pay at most $D + K$: it will have one of the covers as $[jD, (j+1)D]$ and the remaining 1-bits can be covered by $k-1$ intervals of unit length. Now if D is much larger than K, then we get the desired result. ◀

References

1 Hamid A. Toussi, Ahmed Khademzadeh: Improving bit-vector representation of points-to sets using class hierarchy CoRR abs/1108.2683: (2011)
2 Compilers: Principles, Techniques, and Tools Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Addison Wesley 1986.
3 Moses Charikar, Rina Panigrahy: Clustering to Minimize the Sum of Cluster Diameters.
4 S. R. Doddi, M. V. Marathe, S. S. Ravi, D. S. Taylor, P. Widmayer: Approximation algorithms for clustering to minimize the sum of diameters. Proc. 7th Scandinavian Workshop on Algorithm Theory, 2000: 237-250.
5 Dan Gusfield: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press (1997).
6 Mark Nelson, Jean-Loup Gailly: The Data Compression Book. John Wiley & Sons.
7 Rishi Rakesh Sinha, Marianne Winslett: Multi-resolution bitmap indexes for scientific data. ACM Trans. Database Syst. 32(3): 16 (2007)
8 Rishi Rakesh Sinha, Soumyadeb Mitra, Marianne Winslett: Bitmap indexes for large scientific data sets: a case study. IPDPS 2006
9 Preeti Ranjan Panda, M. Balakrishnan, Anant Vishnoi: Compressing Cache State for Postsilicon Processor Debug. IEEE Trans. Computers 60(4): 484-497 (2011)
10 Preeti Ranjan Panda, Anant Vishnoi, M. Balakrishnan: Enhancing post-silicon processor debug with Incremental Cache state Dumping. VLSI-SoC 2010: 55-60
11 Anant Vishnoi, Preeti Ranjan Panda, M. Balakrishnan: Online cache state dumping for processor debug. DAC 2009: 358-363

Maintaining Approximate Maximum Weighted Matching in Fully Dynamic Graphs

Abhash Anand[1], Surender Baswana[2], Manoj Gupta[3], and Sandeep Sen[4]

1 **Indian Institute of Technology, Kanpur**
abhash@cse.iitk.ac.in
2 **Indian Institute of Technology, Kanpur**
sbaswana@cse.iitk.ac.in*
3 **Indian Institute of Technology, New Delhi**
gmanoj@cse.iitd.ernet.in
4 **Indian Institute of Technology, New Delhi**
ssen@iitd.ernet.in

Abstract

We present a fully dynamic algorithm for maintaining approximate maximum weight matching in general weighted graphs. The algorithm maintains a matching $\mathcal{M}$ whose weight is at least $\frac{1}{8}M^*$ where M^* is the weight of the maximum weight matching. The algorithm achieves an expected amortized $O(\log n \log \mathcal{C})$ time per edge insertion or deletion, where $\mathcal{C}$ is the ratio of the weights of the highest weight edge to the smallest weight edge in the given graph.

1998 ACM Subject Classification E.1 [Data Structures]: Graphs and Networks; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems; G.2.2 [Graph Theory]: Graph Algorithms

Keywords and phrases Matching, Dynamic Algorithm, Graph Algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.257

1 Introduction

Let $G = (V, E)$ be an undirected graph on $n = |V|$ vertices and $m = |E|$ edges. Let there be a weight function $w : E \to \mathbb{R}^+$ such that $w(e)$, for any $e \in E$, represents the weight of e. The weight function for a set of edges $M \subseteq E$ is represented by $w(M)$ and is defined as $\sum_{e \in M} w(e)$.

A subset M of E is a "matching" if no vertex of the graph is incident on more than one edge in M. In an unweighted graph, a maximum matching is defined as the maximum cardinality matching (MCM). In a weighted graph, maximum matching is defined as the matching having maximum weight (MWM). For any $\alpha > 1$, a matching is called α-MWM(α-MCM) if it is at least $\frac{1}{\alpha}$ factor of MWM(MCM).

A dynamic graph algorithm maintains a data structure associated with some property (connectivity, transitive closure, matching) of a dynamic graph. The aim of a dynamic graph algorithm is to handle updates in the graph and answer queries associated with the corresponding property. The updates in the graph can be insertion or deletion of edges (V is assumed to be fixed). The dynamic algorithms which handle only insertions are called

* Research supported by the Indo-German Max Planck Center for Computer Science (IMPECS).

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 257–266

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

incremental algorithms and those that can handle only deletions are called decremental algorithms. An algorithm that can handle both insertions and deletions of edges is called a *fully dynamic* algorithm. In this paper, we present a fully dynamic algorithm for maintaining an approximate maximum weight matching.

Previous Results

The fastest known algorithm for finding MCM in general graphs is by Micali and Vazirani[7] that runs in $O(m\sqrt{n})$ time. Their algorithm can be used to compute a matching having size $(1-\epsilon)$ times the size of maximum matching in $O(m/\epsilon)$ time. Mucha and Sankowski[8] designed an algorithm that computes MCM in $O(n^\omega)$, where $\omega < 2.376$ is the exponent of n in the fastest known matrix multiplication algorithm. Relatively, fewer algorithms are known for maintaining matching in a dynamic graph. The first algorithm was designed by Ivkovic and Lloyd[5] with amortized update time $O(n+m)^{0.7072}$. Onak and Rubinfeld[9] presented an algorithm that achieves expected amortized polylogarithmic update time and maintains an α-approximate MCM where α was claimed to be some large constant but not explicitly calculated. Baswana, Gupta and Sen[1] presented a fully dynamic randomized algorithm for maintaining maximal matching in expected amortized $O(\log n)$ update time. It is well known that a maximal matching is a 2-MCM as well.

For computing maximum weight matching Gabow[4] designed an $O(mn+n^2\log n)$ time algorithm. Preis[10] designed a $O(m)$ time algorithm for computing a 2-MWM. Drake and Hougardy[2] designed a simpler algorithm for the same problem. Vinkemeier and Hougardy[11] presented an algorithm to compute a matching which is $(2/3-\epsilon)$ times the size of MWM in $O(m/\epsilon)$ time. Duan, Pettie and Su[3] presented an algorithm to compute a matching which is $(1-\epsilon)$ times the size of MWM in $O(m\epsilon^{-1}\log\epsilon^{-1})$ time. To the best of our knowledge, there have been no sublinear algorithm for maintaining MWM or approximate MWM in dynamic graphs.

Preliminaries

Let M be a matching in a graph $G=(V,E)$. A vertex in the graph is called *free* with respect to M if it is not incident on any edge in M. A vertex which is not free is called *matched*. Similarly, an edge is called *matched* if it is in M and is called *free* otherwise. If (u,v) is a matched edge, then u is called be the *mate* of v and vice versa. A matching M is said to be *maximal* if no edge can be added to the matching without violating the degree bound of one for a matched vertex. An alternating path is defined as a path in which edges are alternately matched and free, while an augmenting path is an alternating path which begins and ends with free vertices.

Our Results

We present a fully dynamic algorithm that achieves expected amortized $O(\log n\log\mathcal{C})$ update time for maintaining 8-MWM. Here $\mathcal{C}$ is the ratio of the weights of the highest weight edge to the smallest weight edge in the given graph. Our algorithm uses, as a subroutine, the algorithm of Baswana, Gupta and Sen [1] for maintaining a maximal matching. We can state the main result of [1] formally in the following theorem.

▶ **Theorem 1.** *Starting from an empty graph on n vertices, a maximal matching in the graph can be maintained over any arbitrary sequence of t insertion and deletion of edges in $O(t\log n)$ time in expectation and $O(t\log n+n\log^2 n)$ time with high probability.*

Note that for the above algorithm, the matching(random bits) at any time is not known to the adversary[1] for it to choose the updates adaptively.

The idea underlying our algorithm has been inspired by the algorithm of Lotker, Patt-Shamir, and Rosen[6] for maintaining approximate MWM in distributed environment. Their algorithm maintains a 27-MWM in a distributed graph and achieves $O(1)$ rounds to update the matching upon any edge insertion or deletion.

Overview of our approach

Given that there exists a very efficient algorithm [1] for maintaining maximal matching (hence 2-MCM), it is natural to explore if this algorithm can be employed for maintaining approximate MWM. Observe that MCM is a special case of MWM with all edges having the same weight. Since a maximal matching is 2-MCM, it can be observed that a maximal matching is 2-MWM in a graph if all its edges have the same weight. But this observation does not immediately extend to the graphs having non-uniform weights on edges. Let us consider the case when the edge weights are within a range, say, $[\alpha^i, \alpha^{i+1})$, where $\alpha > 1$ is a constant. In such a graph the maximal matching gives a 2α approximation of the maximum weight matching. So, a maximal matching can be used as an approximation for MWM in a graphs where the ratio of weights of maximum weight edge to the smallest weight edge is bounded by some small constant. To exploit this observation, we partition the edges of the graph into levels according to their weight. We select a constant $\alpha > 1$ whose value will be fixed later on. Edges at level i have weights in the range $[\alpha^i, \alpha^{i+1})$ and the set of edges at level i is represented by E_i, viz., $\forall e \in E_i, w(e) \in [\alpha^i, \alpha^{i+1})$.

Observe that in this scheme of partitioning, any edge is present only at one level. The subgraph at level i is defined as $G_i = (V, E_i)$. We maintain a maximal matching M_i for G_i using the algorithm of Baswana, Gupta and Sen[1]. The maximal matching at each level provides an approximation for the maximum weight matching at that level. However, $\cup_i M_i$ is not necessary a matching since a vertex may have multiple edges incident on it from $\cup_i M_i$. Let $\mathcal{H} = (V, \bigcup M_i)$ be the subgraph of G having only those edges which are part of the maximal matching at some level. Our algorithm maintains a matching in the subgraph $\mathcal{H}$ which is guaranteed to be 8-MWM for the original graph G. The algorithm builds on the algorithm in [1], though the analysis of algorithm for maintaining 8-MWM is not straightforward.

2 Fully Dynamic 8-MWM

Our algorithm maintains a partition of edges according to their levels. A maximal matching M_i is maintained at each level using the fully dynamic algorithm in [1]. While processing any insertion or deletion of an edge, this algorithm will lead to change in the status of edges from being matched to free and vice-versa. This leads to deletion or insertion of edges from/to $\mathcal{H}$. However, since the algorithm [1] achieves expected amortized $O(\log n)$ time per update, so the expected amortized number of deletions and insertions of edges in $\mathcal{H}$ will also be $O(\log n)$ only. Our algorithm will maintain a matching $\mathcal{M}$ in the subgraph $\mathcal{H}$ taking advantage of the hierarchical structure of $\mathcal{H}$. Since $\mathcal{H}$ is formed by the union of matchings at various levels, a vertex can have at most one neighbor at each level. The matching $\mathcal{M}$ is maintained such that for every edge of $\mathcal{H}$ which is not in $\mathcal{M}$ there must be an edge adjacent

[1] The oblivious adversarial model is also used in randomized data-structure like universal hashing

to it at a higher level which is in $\mathcal{M}$. For an edge e, let $Level(e)$ denote its level. In precise words, the algorithm maintains the following invariant after every update.

$\forall e \in E(\mathcal{H})$, either $e \in \mathcal{M}$ or e is adjacent to an edge $e' \in \mathcal{M}$ such that $Level(e') > Level(e)$.

Notations

The algorithm maintains the following information at each stage.

- M_l - A maximal matching at the level l.
- $Free(v)$ - A variable which is true if v is free in the matching $\mathcal{M}$, and false otherwise.
- $Mate(v)$ - The mate of v, if it is not free.
- $Level((u,v))$ or $Level(e)$ - The level at which the edge e or the edge (u,v) is present according to the condition that $\forall e \in G_i, w(e) \in [\alpha^i, \alpha^{i+1})$.
- $OccupiedLevels$ - The set of levels where there is at least one edge from $\mathcal{H}$.
- L^{max} - The highest occupied level.
- L^{min} - The lowest occupied level.
- $N(v,i)$ - The neighbor of v in M_i, if any, and *null* otherwise.
- $\mathcal{M}$ - The matching maintained by our algorithm.

For a better understanding of our fully dynamic algorithm, the following section describes its static version for computing $\mathcal{M}$ in the graph $\mathcal{H}$.

Static Algorithm to obtain $\mathcal{M}$ from $\mathcal{H}$

Procedure 2.1: StaticCombine()

```
1 M = φ;
2 for i = L^max to L^min do
3     M = M ∪ M_i;
4     for (u,v) ∈ M_i do
5         for j = i − 1 to L^min do
6             for (x,y) ∈ M_j do
7                 if u = x or u = y or v = x or v = y then
8                     M_j = M_j \ {(x,y)};
```

The static algorithm divides the edges of the graph G into levels and a maximal matching M_i is obtained for each of the levels. Using these maximal matchings we get the graph $\mathcal{H}$. Thereafter the level numbers L^{max} and L^{min} are computed and the procedure `StaticCombine` is executed.

The procedure `StaticCombine` starts by picking all the edges in $\mathcal{H}$ at the highest level and adds them to the matching $\mathcal{M}$. For every edge (u,v) added to the matching $\mathcal{M}$, all the edges in the graph $\mathcal{H}$ incident on u and v have to be removed from the graph. The same process is repeated for the next lower level. Note that every edge in $\mathcal{H}$ is either in the matching $\mathcal{M}$ or its neighboring edge at some higher level is in $\mathcal{M}$ and thus the invariant is maintained. Observe that the matching $\mathcal{M}$ is a maximal matching in $\mathcal{H}$ because of the way it is being computed.

Dynamic Algorithm to maintain $\mathcal{M}$

After each insertion or deletion of any edge, our algorithm maintains a matching $\mathcal{M}$ satisfying the invariant described above. Our algorithm processes insertions and deletions of edges in $\mathcal{H}$ to update $\mathcal{M}$. An addition and deletion of the edges in $\mathcal{H}$ is caused due to addition/deletion of an edge in the original graph G. We describe some basic procedures first. Then the procedures for handling addition and deletion of edges in $\mathcal{H}$ are described and finally the procedures for handling addition and deletion of edges in G are described.

Procedure 2.2: AddToMatching(u, v)

```
1 Free(u) = False; Free(v) = False;
2 Mate(u) = v; Mate(v) = u;
3 M = M ⋃ {(u, v)};
```

Procedure 2.3: DelFromMatching(u, v)

```
1 Free(u) = True; Free(v) = True;
2 M = M \ {(u, v)};
```

The procedure `AddToMatching` adds an edge to the matching $\mathcal{M}$ updating the free and mate fields accordingly. The procedure `DelFromMatching` deletes an edge from the matching $\mathcal{M}$ updating the mate and the free fields correctly. Both of them execute in $O(1)$ time.

Procedure 2.4: HandleFree(u, lev)

```
1  for l from lev to L^min do
2      v = N(u, l);
3      if v is not null then
4          if v is free then
5              AddToMatching (u, v);
6              return;
7          else if Level((v, Mate(v))) < l then
8              v' = Mate(v);
9              DelFromMatching (v, v');
10             AddToMatching (u, v);
11             HandleFree (v', Level((v, v')));
12             return;
```

The procedure `HandleFree` takes as an input a vertex u which has become free in $\mathcal{M}$ and a level number lev from where it has to start looking for a mate. Note that it follows from the invariant that u does not have any free neighbor at any level above lev. The procedure `HandleFree` proceeds as follows. It searches for a neighbor of u in the decreasing order of levels starting from lev. In this process, on reaching a level $l \leq lev$ if it finds a free neighbor of u, the corresponding edge is added to the matching $\mathcal{M}$ and the procedure stops. Otherwise if some neighbor v is found which already has a mate at some lower level than l, then notice that we are violating the invariant as (u, v) does not belong to $\mathcal{M}$ and

is neighboring to an edge in $\mathcal{M}$ at a lower level. So, the edge $(v, Mate(v))$ is removed from the matching $\mathcal{M}$, and the edge (u, v) is added to the matching $\mathcal{M}$. This change results in a free vertex which is at a lower level and so we proceed recursively to process it. Note that the recursive calls to `HandleFree` are all with lower level numbers. So, the procedure takes $O(L^{max} - L^{min})$ time.

Procedure 2.5: AddEdge(u, v)

```
1  l = Level((u, v));
2  N(u, l) = v;
3  N(v, l) = u;
4  if u is free and v is free then
5      AddToMatching (u, v);
6  else if u is free and v is not free then
7      if Level((u, Mate(u))) < l then
8          v' = Mate(v);
9          DelFromMatching (v, v');
10         AddToMatching (u, v);
11         HandleFree (v', Level((v, v')));
12 else if u is not free and v is free then
13     if Level((v, Mate(v))) < l then
14         u' = Mate(u);
15         DelFromMatching (u, u');
16         AddToMatching (u, v);
17         HandleFree (u', Level((u, u')));
18 else if Level((v, Mate(v))) < l and Level((u, Mate(u))) < l then
19     u' = Mate(u); v' = Mate(v);
20     DelFromMatching (u, u'); DelFromMatching (v, v');
21     AddToMatching (u, v);
22     HandleFree (u', Level((u, u')));
23     HandleFree (v', Level((v, v')));
```

The procedure `AddEdge` handles addition of edges to $\mathcal{H}$. Suppose the edge (u, v) is added to $\mathcal{H}$. If both u and v are free with respect to $\mathcal{M}$, then the edge (u, v) is added to the matching $\mathcal{M}$. Otherwise, there must be some edge(s) in $\mathcal{M}$ adjacent to (u, v). This follows due to the fact that $\mathcal{M}$ is a maximal matching in $\mathcal{H}$. If (u, v) is adjacent to a higher level edge in $\mathcal{M}$, then nothing is done. If (u, v) is adjacent to some lower level edge(s) in $\mathcal{M}$, then notice that the invariant maintained by the algorithm gets violated. Therefore, we remove these lower level edge(s) (adjacent to (u, v)) from the matching $\mathcal{M}$ and adds the edge (u, v) to the matching. At most 2 vertices can become free due to the addition of this edge to $\mathcal{M}$ and we handle them using the procedure `HandleFree`. If u' was the previous mate of u, then the edge (u, u') is removed from $\mathcal{M}$. Since $\mathcal{M}$ satisfied the invariant before addition of this edge, all the neighboring edges of u' at higher level than $Level(u, u')$ are matched to a vertex at higher levels. So u' has to start looking for mates from the level of (u, u'). The procedure makes a constant number of calls to `HandleFree` and thus runs in $O(L^{max} - L^{min})$ time.

The procedure `DeleteEdge` does nothing if an unmatched edge from $\mathcal{H}$ is deleted. If a matched edge (u, v) is deleted at level l, it calls `HandleFree` for both the end points to

Procedure 2.6: DeleteEdge(u, v)

```
1 l = Level((u, v));
2 N(u, l) = null;
3 N(v, l) = null;
4 if (u, v) ∈ M then
5     DelFromMatching (u, v);
6     HandleFree (u, l);
7     HandleFree (v, l);
```

restore the invariant in the matching. `HandleFree` is called with the level l because our invariant implies that all the neighbors of u and v are matched at higher levels. So they cannot find a mate at higher levels. This again takes $O(L^{max} - L^{min})$ time.

Procedure 2.7: EdgeUpdate$(u, v, type)$

```
1  l = Level((u, v)) = ⌊log_α w(u, v)⌋;
2  if type is addition and M_l is φ then
3      OccupiedLevels = OccupiedLevels ⋃ {l};
4      Update L^max and L^min;
5  Update M_l using the algorithm in [1];
6  if type is deletion and M_l is φ then
7      OccupiedLevels = OccupiedLevels \ {l};
8      Update L^max and L^min;
9  Let D be the set of edges deleted from M_l in step 5;
10 Let A be the set of edges added to M_l in step 5;
11 for (x, y) ∈ D do
12     DeleteEdge (x, y);
13 for (x, y) ∈ A do
14     AddEdge (x, y);
```

The function `EdgeUpdate` handles addition and deletion of an edge in G. It finds out the level of the edge and updates the maximal matching at that level using the algorithm of Baswana, Gupta and Sen [1]. It updates the *OccupiedLevels* set accordingly. This set is required because the values of L^{max} and L^{min} are to be maintained. The algorithm [1] can be easily augmented to return the set of edges being added or deleted from the maximal matching during each update in the graph G. As discussed before, expected amortized $O(\log n)$ edges change their status in $\mathcal{H}$ per update in G. Our algorithm processes these updates in $\mathcal{H}$ as described above. So, overall our algorithm has an expected amortized update time of $O(\log n \cdot (L^{max} - L^{min}))$. Let e^{max} and e^{min} represent the edges having the maximum and the minimum weight in the graph. Recall that $\mathcal{C} = w(e^{max})/w(e^{min})$.

$$L^{max} - L^{min} < \log_\alpha w(e^{max}) - \log_\alpha w(e^{min}) + 1 = O\left(\log \frac{w(e^{max})}{w(e^{min})}\right) = O(\log \mathcal{C})$$

So we can claim that

▶ Claim 2.1. The expected amortized update time of the algorithm per edge insertion or deletion is $O(\log n \log \mathcal{C})$.

In the next section we analyze the algorithm to prove that the matching $\mathcal{M}$ maintained by it at each stage is indeed 8-MWM.

2.1 Analysis

To get a good approximation ratio, we bound the weight of M^* with the weight of $\mathcal{M}$. We now state a few simple observations which help in understanding the analysis.

▶ Observation 2.1. Since M^* is a matching, no two edges of M^* can be incident on the same vertex.

▶ Observation 2.2. For any edge $e \notin M^*$, there can be at most two edges of M^* which are adjacent to e, one for each endpoint of e.

To bound the weight of M^* using the weight of $\mathcal{M}$, we define a many to one mapping $\phi : M^* \to \mathcal{M}$. This mapping maps every edge in M^* to an edge in $\mathcal{M}$. Using this mapping, we find out all the edges which are mapped to an edge $e \in \mathcal{M}$ and bound their weight using the weight of e. Let this set be denoted by $\phi^{-1}(e)$. For an edge $e^* \in M^*$, the mapping is defined as:

1. If $e^* \in E(\mathcal{H})$ and $e^* \in \mathcal{M}$ then $\phi(e^*) = e^*$.
2. If $e^* \in E(\mathcal{H})$ and $e^* \notin \mathcal{M}$ then our invariant ensures that e^* is adjacent to an edge $e \in \mathcal{M}$ such that $\text{Level}(e) > \text{Level}(e^*)$. In this case, we define $\phi(e^*) = e$. If e^* is adjacent to two matched edges in $\mathcal{M}$, map e^* to any one of them. As a rule, if two edges are available for mapping, then we will map e^* to any one of them.
3. If $e^* \notin E(\mathcal{H})$, then consider its level, say i. Since we maintain a maximal matching M_i at level i, at least one of the end point of e^* must be present in M_i. Let $e \in M_i$ be adjacent to e^*. If $e \in \mathcal{M}$, we define $\phi(e^*) = e$.
4. If $e^* \notin E(\mathcal{H})$ and the edge $e \in M_i$ adjacent to e^* is not present in $\mathcal{M}$ then e must be adjacent to an edge $e' \in \mathcal{M}$ such that $\text{Level}(e') > \text{Level}(e)$. In this case, we define $\phi(e^*) = e'$.

An edge $e^* \in M^*$ can either be present or absent from $E(\mathcal{H})$. If it is in $E(\mathcal{H})$, then it is mapped using type 1 and type 2 mapping else it is mapped using type 3 and type 4 mapping. This implies all the edges in M^* are mapped by ϕ. Now that we have defined a many to one mapping, we find out the edges of M^* which are mapped to an edge $e \in \mathcal{M}$. An edge which is mapped to an edge $e \in \mathcal{M}$ can either be e itself or be adjacent to e or not adjacent to e. If an edge of M^*, which is mapped to $e \in \mathcal{M}$, is e itself or is adjacent to e, then it is called a *Directly mapped edge*. An edge of M^* which is mapped to $e \in \mathcal{M}$ and is not adjacent to e is called an *Indirectly mapped edge*. Let $\phi_D^{-1}(e)$ and $\phi_I^{-1}(e)$ be the set of edges from M^* mapped directly and indirectly respectively to an edge $e \in \mathcal{M}$. Directly mapped edges are of type 1, 2 and 3 and indirectly mapped edges are of type 4.

If an edge $e \in \mathcal{M}$ has an edge of type 1 directly mapped to it, then e will not have any other edge directly mapped to it. This follows from the definition of a directly mapped edge and Observation 2.1. There can be at most two directly mapped edges of the type 2 (Observation 2.2). These edges mapped to e are always from a level $< Level(e)$. There can be at most two directly mapped edges of type 3 also if they are not in $\mathcal{H}$ but are adjacent to e. By Observation 2.2, there can only be two such edges.

▶ Claim 2.2. For an edge $e \in \mathcal{M}$, there can be at most two edges from M^* that are directly mapped to e.

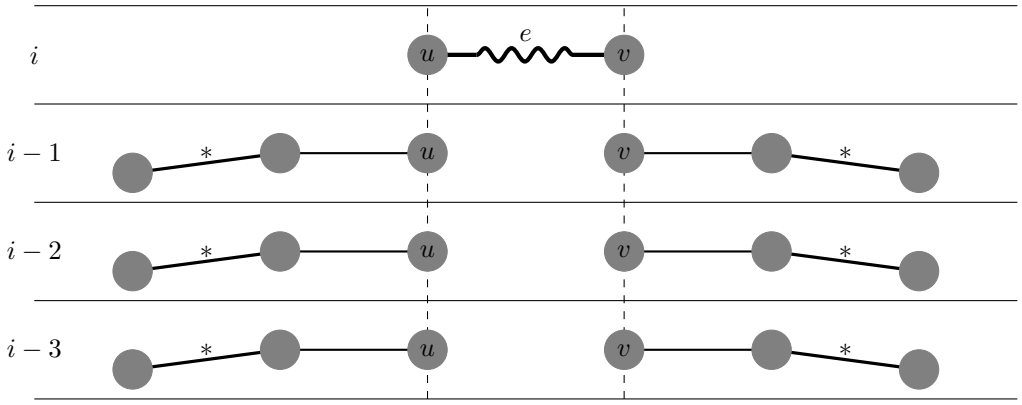


Figure 1 $e \in \mathcal{M}$. The edges marked $*$ are not in $\mathcal{H}$ and are in M^*. The edges which are not marked $*$ are all in $\mathcal{H}$. All the edges marked by $*$ are indirectly mapped to e.

The total weight of the edges directly mapped to e will be maximum when both of them are from the same level as e. Assume that e is at level i. Summing the weights of the edges which are directly mapped to e, we get

$$\sum_{e^* \in \phi_D^{-1}(e)} w(e^*) < 2 * \alpha^{i+1} < 2\alpha w(e) \tag{1}$$

An edge $e^* \in M^*$ mapped indirectly to an edge $e \in \mathcal{M}$ can only be of type 4. In this case e^* is not in $\mathcal{H}$, but is adjacent to an edge in $\mathcal{H}$, which in turn is adjacent to e. By definition, these edges are from a level lower than that of e. There can be at most two edges from each level lower than $Level(e)$ which are in $\mathcal{H}$ and are adjacent to e(see Figure 1).

▶ Claim 2.3. There can be at most two indirectly mapped edges to an edge $e \in \mathcal{M}$ at level $< Level(e)$.

Note that there can be a large number of edges which are indirectly mapped to e. Still we will be able to get a good bound on their total weight. This is because there can be at most two indirectly mapped edges from each level and the weight of edges in the levels decreases geometrically as we go to lower levels.

Assume that e is at level i. Summing the weight of edges which are indirectly mapped to e, we get

$$\sum_{e^* \in \phi_I^{-1}(e)} w(e^*) < 2 \sum_{j=i-1}^{L^{min}} \alpha^{j+1} < \frac{2\alpha^{i+1}}{\alpha - 1} < \frac{2\alpha w(e)}{\alpha - 1} \tag{2}$$

Thus, the total weight mapped to e is -

$$\sum_{e^* \in \phi^{-1}(e)} w(e^*) = \sum_{e^* \in \phi_D^{-1}(e)} w(e^*) + \sum_{e^* \in \phi_I^{-1}(e)} w(e^*) < w(e) \left(\frac{2\alpha}{\alpha - 1} + 2\alpha \right)$$

As reasoned before, an edge in M^* is mapped to some edge in $\mathcal{M}$. So summing this over all the edges in $\mathcal{M}$, we get

$$\sum_{e \in M} w(e) \left(\frac{2\alpha}{\alpha - 1} + 2\alpha \right) > \sum_{e \in M} \sum_{e^* \in \phi^{-1}(e)} w(e^*) = \sum_{e^* \in M^*} w(e^*)$$

The function $f(\alpha) = \left(\frac{2\alpha}{\alpha-1} + 2\alpha\right)$ attains its minimum value of 8 at $\alpha = 2$. So, if the value of α is picked to be 2, we get an 8 approximate maximum weight matching algorithm. We can state the following theorem.

▶ **Theorem 2.** *There exists a fully dynamic algorithm that maintains 8-MWM for any graph on n vertices in expected amortized $O(\log n \log \mathcal{C})$ time per update.*

3 Conclusion

We presented a fully dynamic algorithms for maintaining matching of large size or weight in graphs. The algorithm maintains a 8-MWM with expected $O(\log n \log \mathcal{C})$ amortized update time.

References

1 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in O(log n) update time. In *52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 383–392. IEEE, 2011.

2 D.E. Drake and S. Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85(4):211–213, 2003.

3 R. Duan, S. Pettie, and H.H. Su. Scaling algorithms for approximate and exact maximum weight matching. *Arxiv preprint arXiv:1112.0790*, 2011.

4 H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443. Society for Industrial and Applied Mathematics, 1990.

5 Z. Ivkovic and E. Lloyd. Fully dynamic maintenance of vertex cover. In *Graph-Theoretic Concepts in Computer Science*, pages 99–111. Springer, 1994.

6 Z. Lotker, B. Patt-Shamir, and A. Rosen. Distributed approximate matching. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 167–174. ACM, 2007.

7 S. Micali and V.V. Vazirani. An O($\sqrt{|V|}$|E|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science*, pages 17–27. IEEE, 1980.

8 M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255. IEEE, 2004.

9 K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 457–464. ACM, 2010.

10 R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *STACS 99*, pages 259–269. Springer, 1999.

11 D.E.D. Vinkemeier and S. Hougardy. A linear-time approximation algorithm for weighted matchings in graphs. *ACM Transactions on Algorithms (TALG)*, 1(1):107–122, 2005.

Approximation Algorithms for the Unsplittable Flow Problem on Paths and Trees

Khaled Elbassioni[1], Naveen Garg[2], Divya Gupta[3], Amit Kumar[2], Vishal Narula[4], and Arindam Pal[2]

1 **MPI-Informatik, Germany**
2 **IIT Delhi, India**
3 **UCLA, USA**
4 **Goldman Sachs, India**

Abstract

We study the Unsplittable Flow Problem (UFP) and related variants, namely UFP with Bag Constraints and UFP with Rounds, on paths and trees. We provide improved constant factor approximation algorithms for all these problems under the *no bottleneck assumption* (NBA), which says that the maximum demand for any source-sink pair is at most the minimum capacity of any edge. We obtain these improved results by expressing a feasible solution to a natural LP relaxation of the UFP as a near-convex combination of feasible integral solutions.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, Integer Decomposition, Linear Programming, Scheduling, Unsplittable Flows

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.267

1 Introduction

In this paper, we give new results for several variants of the Unsplittable Flow Problem on paths and trees. The setting for all of these problems is as follows: we are given a graph $G = (V, E)$, where G is either a path or a tree, with edge capacities c_e for each edge $e \in E$, a set of demands $D_1, \ldots, D_m$, where each demand D_i consists of a source-sink pair s_i, t_i, a bandwidth requirement d_i, and a profit w_i. In order to route a demand D_i, we send d_i amount of flow from s_i to t_i along the (unique) path between them in G. A set of demands is said to be *feasible* if they can be simultaneously routed without violating any edge capacity. In the Max-UFP problem, we would like to find a feasible subset of demands of maximum total profit. In the Round-UFP problem, we would like to color the demands with minimum number of colors such that demands with a particular color form a feasible subset. Another interesting variant is the Bag-UFP problem, where we are given sets $\mathcal{D}_1, \ldots, \mathcal{D}_k$, where each set (or bag) $\mathcal{D}_i$ consists of a set of demands, and has an associated profit (the individual demands in each set do not have profits, though they could have different bandwidth requirements). A solution needs to pick at most one demand from each bag such that these demands are feasible. The goal is to maximize the total profit of bags from which a demand is picked.

All of the above problems are NP-Hard (even for a path), and there has been lot of recent work on obtaining constant factor approximation algorithms for them. An assumption often made in these settings is the so-called *no-bottleneck assumption*: the maximum bandwidth requirement of any demand is at most the minimum edge capacity, i.e., $\max_i d_i \leq \min_e c_e$. Obtaining constant factor approximation algorithms for the above problems without the

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 267–275

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

no-bottleneck assumption remains a challenging task; the only exception being the recent result of Bonsma et al. [4] which gives a constant factor approximation algorithm for MAX-UFP on the line. We will assume that the *no-bottleneck assumption* holds in subsequent discussions.

MAX-UFP and BAG-UFP are weakly NP-Hard, since they contain the KNAPSACK problem as a special case, where there is just a single edge. Recently, it has been proved that the problem is strongly NP-hard, even for the restricted case where all demands are chosen from $\{1, 2, 3\}$ and all capacities are uniform [4]. However, the problem is not known to be APX-hard, so a polynomial time approximation scheme (PTAS) may still be possible. For the special case of (KNAPSACK), an FPTAS is well-known. When all capacities, demands and profits are 1, MAX-UFP specializes to MAX-EDP, the maximum edge-disjoint paths problem.

Chakrabarti et al. [7] gave the first constant approximation algorithm for MAX-UFP on paths and the approximation ratio was subsequently improved to $2 + \varepsilon$, for any constant $\varepsilon > 0$ by Chekuri et al. [8]. They also gave a constant factor approximation algorithm for MAX-UFP on trees. These algorithms are based on the idea of rounding a natural LP relaxation of the MAX-UFP problem.

ROUND-UFP is NP-Hard, since it contains the BIN PACKING problem as a special case, where there is just a single edge. BIN PACKING is known to be APX-hard. However, it has an asymptotic polynomial time approximation scheme (APTAS). There are also simple greedy algorithms like *first-fit* and *best-fit*, which give constant-factor approximations (see e.g. [1]). When all capacities and demands are 1, ROUND-UFP reduces to the interval coloring problem on paths, for which a simple greedy algorithm gives the optimal coloring.

The ROUND-UFP problem for paths has been well-studied in the context of on-line algorithms as well. Here the intervals arrive in arbitrary order, and we need to assign them a color on their arrival so that all intervals with one color form a feasible packing, i.e. total demand on any edge does not exceed its capacity. When all capacities and demands are 1, i.e. when no two intersecting intervals can be given the same color, the first-fit algorithm achieves a constant competitive ratio ([15, 16, 18]). Kierstead and Trotter [14] gave a different online algorithm which uses at most $3\omega - 2$ colors. They also proved that any deterministic online algorithm in the worst case will require at least $3\omega - 2$ colors. For the case of arbitrary edge capacities and demands with the no-bottleneck assumption (NBA), which is same as ROUND-UFP, Epstein et al. [12] gave a 78-competitive algorithm. Prior to our work, this was the best known result for ROUND-UFP in the off-line setting as well.

The BAG-UFP problem was introduced by Chakaravarthy et al. [6], who gave an $O\left(\log\left(\frac{c_{\max}}{c_{\min}}\right)\right)$-approximation algorithm. Here $c_{\max}$ and $c_{\min}$ are the maximum and minimum edge capacities of the path respectively. They gave the first constant factor approximation algorithm for the BAG-UFP problem on paths – the approximation ratio is 120. A related problem is the *job interval selection* problem for which Chuzhoy et al. [11] gave an $\left(\frac{e}{e-1}\right)$-approximation algorithm. See also Erlebach et al. [13] for some additional results.

1.1 Our Contributions

In this paper, we give a unified framework for these problems. We give a simple algorithm for ROUND-UFP on paths. We use this to give a constant factor approximation algorithm for MAX-UFP as well. The idea is to start with a natural LP relaxation for MAX-UFP. We show that using our algorithm for ROUND-UFP, one can express a fractional solution to the LP as a convex combination of integer solutions (up to a constant factor). This

idea generalizes to Bag-UFP as well. This leads to improved approximation algorithms for several of these problems. More specifically, our results are:

- We give a 24-approximation algorithm for Round-UFP on paths. This is much simpler than the 78-competitive algorithm of [12], and gives an improved approximation ratio.
- We give a 65-approximation algorithm for Bag-UFP on paths, thus improving the constant approximation factor of 120 given by Chakaravarthy et al. [5] for this problem.
- For trees, we give the first constant factor approximation algorithm for Round-UFP – the approximation factor is 64.

1.2 Other Related Work

Recently, Bonsma et al. [4] gave the first constant factor approximation algorithm for Max-UFP on a path without assuming NBA. They also proved that the problem is strongly NP-hard, even for the restricted case where all demands are chosen from $\{1, 2, 3\}$ and all capacities are uniform.

The round version of Bag-UFP is hard to approximate, because *scheduling jobs with interval constraints* is a special case of this problem. In the latter problem, we have a collection of jobs, where each job has a set of intervals associated with it. We can schedule a job in any of the intervals from its set. The goal is to color the jobs with minimum number of colors, such that the set of jobs with a particular color are feasible, i.e., one can pick an interval from the set associated with each job, such that these intervals are disjoint. Chuzhoy et al. [10] proved that it is NP-hard to get better than $O(\log \log n)$-approximation algorithm for this problem. In the continuous version of this problem, the intervals associated with a job form a continuous time segment, described by a release date and a deadline. Chuzhoy and Codenotti [9] gave a constant factor approximation algorithm for the continuous version.

1.3 Organization of the Paper

In Section 2, we define the problems considered in this paper. We give a constant factor approximation algorithm for Round-UFP on paths in Section 3. In Section 4, we use the ideas developed in Section 3 to get a constant factor approximation algorithm for Max-UFP on paths, which we then extend to Bag-UFP on paths in Section 5. In Section 6 we give a constant factor approximation algorithm for Round-UFP on trees.

2 Preliminaries

We formally define the problems considered in this paper. In all of these problems, an instance will consist of a graph $G = (V, E)$, which is either a path or a tree, with edge capacities c_e for all edges $e \in E$. In case of Round-UFP and Max-UFP, we are also given a set of demands $D_1, \ldots, D_m$. Demand D_i has an associated source-sink pair, (s_i, t_i), a bandwidth requirement d_i and a profit w_i. We shall use I_i to denote the associated unique path between s_i and t_i in G (in case of a path, we shall also call this an interval). A subset of demands will be called *feasible* if they can be routed without violating the edge capacities. In Max-UFP, the goal is to find a feasible subset of demands of maximum total profit. In Round-UFP, the goal is to partition the set of demands into minimum number of colors, such that demands with a particular color are feasible.

Finally, we define the Bag-UFP problem. We will consider this problem for the case of paths only. Here, we are given sets, which we will call *bags*, $\mathcal{D}^1, \ldots, \mathcal{D}^k$, where each set

$\mathcal{D}^j$ consists of a set of demands $D_1^j, \ldots, D_{n_j}^j$. As before, each demand D_i^j is specified by an interval I_i^j and a bandwidth requirement d_i^j. We are also given profits p^j associated with each of the bags $\mathcal{D}^j$. A feasible solution to such an instance picks at most one demand from each of the bags – the selected demands should form a feasible set of routable demands. The profit of such a solution is the total profit of the bags from which we select a demand. The goal is to maximize the total profit.

We require our instances to satisfy the so called *no-bottleneck assumption*. This assumption states that $\max_i d_i \leq \min_e c_e$, where i varies over all the demands, and e varies over all the edges in G. We now give some definitions which will be used by all the algorithms. We will divide the set of demands into two classes – large and small demands.

▶ **Definition 1.** The *bottleneck capacity* b_i of a demand D_i is the smallest capacity of an edge in the (unique) path between s_i and t_i – such an edge is called the *bottleneck edge* for demand D_i. A demand D_i is said to be *small* if $d_i \leq \frac{1}{4}b_i$, else it is a *large* demand.

3 Approximation Algorithm for Round-UFP

We consider an instance $\mathcal{I}$ of the Round-UFP problem given by a path G on n points, and a set of demands $D_1, \ldots, D_m$ as described in Section 2. Let $\mathcal{O}$ denote an optimal solution, and $\texttt{col}(\mathcal{O})$ denote the number of colors used by $\mathcal{O}$. We begin with a few definitions.

▶ **Definition 2.** The *congestion* of an edge e, r_e, is defined as $\left\lceil \frac{\sum_{i:e\in I_i} d_i}{c_e} \right\rceil$, i.e., the ratio of the total demand through the edge e to its capacity. Let $r_{\max} = \max_e r_e$ be the *maximum congestion* on the path.

▶ **Definition 3.** The *clique size* on an edge e, L_e, is defined as the number of large demands containing the edge e. Let $L_{\max} = \max_e L_e$ be the *maximum clique size* on the path.

Clearly, $\texttt{col}(\mathcal{O}) \geq r_{\max}$. We give an algorithm $\mathcal{A}$ which uses $O(r_{\max})$ colors. This will give a constant factor approximation algorithm for this problem. We first consider the case of large demands.

▶ **Lemma 4.** *We can color all large demands with at most $L_{\max}$ colors. Further, $L_{\max} \leq 8\texttt{col}(\mathcal{O})$.*

Proof. We will use the following result of Nomikos et al. [17].

▶ **Lemma 5.** *[17] Consider an instance of* Round-UFP *where all capacities are integers and all demands D_i have bandwidth requirement $d_i = 1$. Then, one can color these demands with $r_{\max}$ colors.*

We first scale all capacties and demand requirements such that $c_{\min}$ becomes equal to 1. Now, we round all capacities down to the nearest integer, and we scale all the demand requirements d_i to 1. Note that this will affect the congestion of an edge e by a factor of at most 8 – since c_e was at least 1, rounding it down to the nearest integer will reduce it by a factor of at most 1/2. Since all demands were of size at least 1/4 (because they are large demands), we may increase the requirement of a demand by factor of at most 4. Thus, the value of $r_{\max}$ will increase by factor of at most 8. Now, we invoke the result in Lemma 5. This proves the lemma. ◀

We now consider the more non-trivial case of small demands. We divide the edges into classes based on their capacities. We say that an edge e is of *class* l if $2^l \le c_e < 2^{l+1}$. We use $\texttt{cl}(e)$ to denote the class of e. For a demand D_j, let l_j be the smallest class such that the interval I_j contains an edge of class l_j. The *critical edge* of demand D_j is defined as the first edge (as we go from left to right from s_j to t_j) in I_j of class l_j. Note that the critical edge could be different from the bottleneck edge, though both of them would be of class l_j.

▶ **Lemma 6.** *The small demands can be colored with at most* $16r_{\max}$ *colors.*

Proof. We maintain $16r_{\max}$ different solutions to the instance $\mathcal{I}$, where a solution routes a subset of the demands. We will be done if we can assign each demand to one of these solutions. Let us call these solutions $\mathcal{S}_1, \ldots, \mathcal{S}_K$, where $K = 16r_{\max}$. We first describe the routing algorithm and then show that it has the desired properties.

We arrange the demands in order of their left end-points – let this ordering be $D_1, \ldots, D_m$. Let e_j be the critical edge of D_j. When we consider D_j, we send it to a solution $\mathcal{S}_l$ for which the total requirements of demands containing e_j is at most $c_{e_j}/16$. At least one such solution must exist, otherwise $r_e > \frac{16r_{\max} \cdot c_{e_j}/16}{c_{e_j}} = r_{\max}$, a contradiction. This completes the description of how we assign each demand to one of the solutions. We now prove that each of the solutions $\mathcal{S}_l$ is feasible.

Fix a solution $\mathcal{S}_l$ and an edge e. Suppose e is of class i. Let $\mathcal{D}(\mathcal{S}_l)$ be the demands routed in $\mathcal{S}_l$ which contain the edge e. Among such demands, let D_u be the last demand for which the critical edge is to the left of e (including e) – let e' be such an edge. Clearly, $\texttt{cl}(e') \ge i$. For an integer $i' \le i$, let $e^{(i')}$ be the first edge of class i' to the right of e (so, $e^{(i)}$ is same as e).

First consider the demands in $\mathcal{D}(\mathcal{S}_l)$ which are considered before (and including D_u). All of these demands go through e' (because all such demands begin before D_u does and contain e). So, the total requirement of such demands, excluding D_u, is at most $c_{e'}/16$ – otherwise we would not have assigned D_u to this solution. Because D_u is a small demand and $\texttt{cl}(e') \ge i$, the total requirements of such demands (including D_u) is at most

$$\frac{2^{i+1}}{16} + \frac{c_e}{4} \le \frac{c_e}{8} + \frac{c_e}{4} = \frac{3c_e}{8}.$$

Now consider the demands in $\mathcal{D}(\mathcal{S}_l)$ whose critical edges are to the right of e – note that, such an edge must be one of $e^{(i')}$ for some $i' < i$. Similar to the argument above, the total requirements of such demands is at most

$$\sum_{i'<i} \left(\frac{2^{i'+1}}{16} + \frac{2^{i'+1}}{4} \right) \le \frac{5 \cdot 2^{i+1}}{16} \le \frac{5c_e}{8}.$$

Thus, we see that the total requirements of demands in $\mathcal{D}(\mathcal{S}_l)$ is at most

$$\frac{5c_e}{8} + \frac{3c_e}{8} \le c_e.$$

Hence the solution is feasible. This proves the lemma. ◀

Combining the above two lemmas, we get the following theorem.

▶ **Theorem 7.** *Given an instance of* Round-UFP*, there is an algorithm for this problem which uses at most* $24 \cdot \texttt{col}(\mathcal{O})$ *colors, and hence it is a* 24*-approximation algorithm. Further, if all demands are small, then one can color the demands using at most* $16 \cdot \texttt{col}(\mathcal{O})$ *colors.*

4 Approximation Algorithms for Max-UFP

In this section we show how ideas from ROUND-UFP can be used to derive a constant factor approximation algorithm for MAX-UFP. Consider an instance $\mathcal{I}$ of MAX-UFP. As before, we divide the demands into small and large demands. For large demands, Chakrabarti et al. [7] showed that one can find the optimal solution by dynamic programming.

▶ **Lemma 8.** *[7] The number of δ-large demands crossing any edge in a feasible solution is at most $\frac{2}{\delta}\left(\frac{1}{\delta}-1\right)$. Hence, an optimum solution can be found in $n^{O(1/\delta^2)}$ time using dynamic programming.*

Note that, according to our definition, large demands are $\frac{1}{4}$-large. Now we consider the small demands. The following lemma gives an approximation algorithm for small demands.

▶ **Lemma 9.** *If there are only small jobs, then there is a* 16*-approximation algorithm for* MAX-UFP.

Proof. We write the following natural LP relaxation for this problem – a variable x_i for demand D_i which is 1 if we include it in our solution, and 0 otherwise.

$$\max \sum_i w_i x_i$$

$$\sum_{i:e\in I_i} d_i x_i \le c_e \quad \text{for all edges } e \tag{1}$$

$$0 \le x_i \le 1 \quad \text{for all demands } i$$

Let $x^\star$ be an optimal solution to the LP relaxation. Let K be an integer such that all the variables $x_i^\star$ can be written as $\frac{\alpha_i}{K}$ for some integer α_i. Now we construct an instance $\mathcal{I}'$ of ROUND-UFP as follows. For each (small) demand D_i in $\mathcal{I}$, we create α_i copies of it. Rest of the parameters are same as those in $\mathcal{I}$. First observe that inequality (1) implies that $\sum_{i:e\in I_i} d_i\alpha_i \le Kc_e, \forall e \in E$. Thus, the congestion of each edge in $\mathcal{I}'$ is at most K. Using Lemma 6 for small demands, we can color the demands with at most $16K$ colors. It follows that the best solution among these $16K$ solutions will have profit at least $\frac{1}{16}\cdot\sum_i w_i x_i^\star$. ◀

Thus, we get the following theorem.

▶ **Theorem 10.** *There is a* 17*-approximation algorithm for the* MAX-UFP *problem.*

Proof. Given an instance $\mathcal{I}$, we divide the demands into large and small demands. For large demands, we compute the optimal solution using Lemma 8, whereas for small demands we compute a solution with approximation ratio 16 using Lemma 9. Then we pick the better of the two solutions.

Consider an optimal solution $\mathcal{O}$ with profit $\texttt{profit}(\mathcal{O})$. Let $\texttt{profit}^l(\mathcal{O})$ be the profit for large demands and $\texttt{profit}^s(\mathcal{O})$ be the profit for small demands. If $\texttt{profit}^l(\mathcal{O}) \ge \frac{1}{17}\cdot\texttt{profit}(\mathcal{O})$, then our solution for large demands will also be at least $\frac{1}{17}\cdot\texttt{profit}(\mathcal{O})$. Otherwise, $\texttt{profit}^s(\mathcal{O}) \ge \frac{16}{17}\cdot\texttt{profit}(\mathcal{O})$. In this case, our solution for small demands will have value at least $\frac{1}{16}\cdot\frac{16}{17}\cdot\texttt{profit}(\mathcal{O}) = \frac{1}{17}\cdot\texttt{profit}(\mathcal{O})$. ◀

5 Approximation Algorithms for Bag-UFP

We now extend the above algorithm to the BAG-UFP problem. Consider an instance $\mathcal{I}$ of this problem. As before, we classify each of the demands D_i^j as either large or small. For each bag, $\mathcal{D}^j$, let $\mathcal{D}^{j,\mathtt{l}}$ be the set of large demands in $\mathcal{D}^j$ and $\mathcal{D}^{j,\mathtt{s}}$ be the set of small demands in $\mathcal{D}^j$. Again, we have two different strategies for large and small demands.

▶ **Lemma 11.** *If there are only large jobs, then there is a* 48*-approximation algorithm for* Bag-UFP*.*

Proof. Suppose we have the further restriction that the selected intervals need to be disjoint. Lemma 8 implies that this will worsen the objective value by a factor of at most 24. However, for the latter problem, we can use the 2-approximation algorithm of Berman et al. [3] and Bar-Noy et al. [2]. This gives a 48-approximation algorithm. ◀

▶ **Lemma 12.** *If there are only small jobs, then there is a* 17*-approximation algorithm for* Bag-UFP*.*

Proof. As in the case of Max-UFP problem, we first write an LP relaxation, and then use an algorithm similar to the one used for the Round-UFP problem. We have a variable x_i^j for demand D_i^j, which is 1 if we include it in our solution and 0 otherwise, and a variable y^j which is 1 if we choose a demand from the bag $\mathcal{D}^j$ and 0 otherwise. The LP relaxation is as follows.

$$\max \sum_j p^j y^j$$

$$\sum_{i:e\in I_i^j} d_i^j x_i^j \le c_e \quad \text{for all edges } e \tag{2}$$

$$\sum_i x_i^j \le y^j \quad \text{for all bags } \mathcal{D}^j \tag{3}$$

$$0 \le x_i^j \le 1 \quad \text{for all demands } i$$

$$0 \le y^j \le 1 \quad \text{for all bags } \mathcal{D}^j$$

Let x, y be an optimal solution to the LP above. Again, let K be a large enough integer such that $y^j = \frac{\alpha^j}{K}, x_i^j = \frac{\beta_i^j}{K}$, where α_j and β_i^j are integers for all j and i. Now we consider an instance of Round-UFP where we have β_i^j copies of the demand D_i^j. The only further restriction is that no two demands from the same bag can get the same color. Inequality (2) implies that $\sum_{i:e\in I_i^j} d_i^j \beta_i^j \le K c_e, \forall e \in E$. So the congestion bound is K. We proceed as in the proof of Lemma 6, except that now we have $17K$ different solutions. When we consider the demand D_i^j, we ignore the solutions which contain a demand from the bag $\mathcal{D}^j$. Inequality (3) implies that $\sum_i \beta_i^j \le \alpha^j \le K, \forall j$. Hence, there will be at most K such solutions. For the remaining $16K$ solutions, we argue as in the proof of Lemma 6. ◀

▶ **Theorem 13.** *There is a* 65*-approximation algorithm for the* Bag-UFP *problem.*

Proof. This follows from the two previous lemmas. We argue as in the proof of Theorem 10. ◀

6 Approximation Algorithms for Round-UFP on Trees

We now consider the Round-UFP problem on trees. Consider an instance $\mathcal{I}$ of this problem as described in Section 2. We consider the case of large and small demands separately. Let $\mathcal{D}^{\mathsf{l}}$ be the set of large demands and D^{s} be the set of small demands.

▶ **Lemma 14.** *There is a* 32*-approximation algorithm for the above instance when we only have demands in* $\mathcal{D}^{\mathsf{l}}$*.*

Proof. Chekuri et al. [8] gave a 4-approximation algorithm for coloring a set of demands when all demands have requirement 1, and the capacities are integers. In fact, their algorithm uses at most $4r_{\max}$ colors. We can reduce our problem to this case by losing an extra factor of 8 in $r_{\max}$ – we proceed exactly as in the proof of Lemma 4. ◀

▶ **Lemma 15.** *There is a* 32*-approximation algorithm for the above instance when we only have demands in* $\mathcal{D}^{\mathbf{s}}$*.*

Proof. The proof is very similar to that of Lemma 6. We maintain $16r_{\max}$ solutions. For a demand D_i, let a_i denote the least common ancestor of s_i and t_i. We consider the demands in a bottom-up order of a_i. For a demand D_i, we define two critical edges: the s_i-critical edge is the critical edge on the $a_i - s_i$ path, and the t_i-critical edge is the critical edge on the $a_i - t_i$-path. We send D_i to the solution in which both these critical edges have been used till $\frac{1}{16}$ of their total capacity only. Again it is easy to check that such a solution will exist. The rest of the argument now follows as in the proof of Lemma 6. ◀

Combining the above two lemmas, we get

▶ **Theorem 16.** *There is a* 64*-approximation algorithm for the* Round-UFP *problem on trees.*

7 Conclusion and Open Problems

In this paper, we studied the Unsplittable Flow Problem and some of its variants, such as UFP with Bag Constraints and UFP with Rounds. We gave improved constant factor approximation algorithms for all these problems under the *no bottleneck assumption*. One important open question is, can we improve the approximation factors further? A related question is, are there lower bounds (hardness results, bad examples or integrality gap examples) for these problems matching these upper bounds? Another important open problem is the approximability of these problems without NBA. For Max-UFP on paths, a $(7 + \varepsilon)$-approximation is known, but for the other problems the question is not settled.

References

1 *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

2 Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Approximating the throughput of multiple machines under real-time scheduling. In *STOC*, pages 622–631, 1999.

3 Piotr Berman and Bhaskar DasGupta. Improvements in throughout maximization for real-time scheduling. In *STOC*, pages 680–687, 2000.

4 Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant factor approximation algorithm for unsplittable flow on paths. In *FOCS*, pages 47–56, 2011.

5 Venkatesan T. Chakaravarthy, Anamitra R. Choudhury, and Yogish Sabharwal. A near-linear time constant factor algorithm for unsplittable flow problem on line with bag constraints. In *FSTTCS*, pages 181–191, 2010.

6 Venkatesan T. Chakaravarthy, Vinayaka Pandit, Yogish Sabharwal, and Deva P. Seetharam. Varying bandwidth resource allocation problem with bag constraints. In *IPDPS*, pages 1–10, 2010.

7 Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47(1):53–78, 2007.

8 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3(3), 2007.

9 Julia Chuzhoy and Paolo Codenotti. Resource minimization job scheduling. In *APPROX-RANDOM*, pages 70–83, 2009.

10 Julia Chuzhoy and Joseph Naor. New hardness results for congestion minimization and machine scheduling. *J. ACM*, 53(5):707–721, 2006.

11 Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Math. Oper. Res.*, 31(4):730–738, 2006.

12 Leah Epstein, Thomas Erlebach, and Asaf Levin. Online capacitated interval coloring. *SIAM J. Discrete Math.*, 23(2):822–841, 2009.

13 Thomas Erlebach and Frits C. R. Spieksma. Interval selection: Applications, algorithms, and lower bounds. *J. Algorithms*, 46(1):27–53, 2003.

14 H.A. Kierstead and W.T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

15 Hal A. Kierstead. The linearity of first-fit coloring of interval graphs. *SIAM J. Discrete Math.*, 1(4):526–530, 1988.

16 Hal A. Kierstead and Jun Qin. Coloring interval graphs with first-fit. *Discrete Mathematics*, 144(1-3):47–57, 1995.

17 Christos Nomikos, Aris Pagourtzis, and Stathis Zachos. Routing and path multicoloring. *Inf. Process. Lett.*, 80(5):249–256, 2001.

18 Sriram V. Pemmaraju, Rajiv Raman, and Kasturi R. Varadarajan. Max-coloring and online coloring with bandwidths on interval graphs. *ACM Transactions on Algorithms*, 7(3):35, 2011.

Graphs, Rewriting and Pathway Reconstruction for Rule-Based Models

Vincent Danos[3], Jérôme Feret[4], Walter Fontana[5], Russell Harmer[1], Jonathan Hayman[4,2], Jean Krivine[1], Chris Thompson-Walsh[2], and Glynn Winskel[2]

1 **CNRS & Université Paris-Diderot, Paris, France**
2 **Computer Laboratory, University of Cambridge, UK**
3 **LFCS, School of Informatics, University of Edinburgh, UK**
4 **LIENS (INRIA/ÉNS/CNRS), Paris, France**
5 **Harvard Medical School, Boston, Massachusetts**

Abstract

In this paper, we introduce a novel way of constructing concise causal histories (pathways) to represent how specified structures are formed during simulation of systems represented by rule-based models. This is founded on a new, clean, graph-based semantics introduced in the first part of this paper for Kappa, a rule-based modelling language that has emerged as a natural description of protein-protein interactions in molecular biology [1]. The semantics is capable of capturing the whole of Kappa, including subtle side-effects on deletion of structure, and its structured presentation provides the basis for the translation of techniques to other models. In particular, we give a notion of trajectory compression, which restricts a trace culminating in the production of a given structure to the actions necessary for the structure to occur. This is central to the reconstruction of biochemical pathways due to the failure of traditional techniques to provide adequately concise causal histories, and we expect it to be applicable in a range of other modelling situations.

1998 ACM Subject Classification F.1.1 Models of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs, J.3 Biology and genetics, I.6.6 Simulation Output Analysis

Keywords and phrases concurrency, rule-based models, graph rewriting, pathways, causality

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.276

1 Introduction

Kappa [8] has emerged as a powerful tool in modelling biochemical systems, supporting sophisticated and efficient simulation [5] and static analysis [6] techniques. It is centered around *rules* that describe how links between sites on entities called agents are modified when local conditions on the link structure (or, more generally, the state of sites) are satisfied. In biological applications, agents typically represent proteins and links correspond to non-covalent associations between domains (sites) of proteins; rules then are intended to capture empirically sufficient conditions for modifications in the binding (or other) state of protein molecules. The use of rule-based approaches has the potential to make a profound impact in these fields [1], making it possible to analyze systems that would be intractable using traditional systems of ordinary differential equations.

Kappa has an intuitive graphical interpretation. In this paper, we formalise the structures involved and characterize the rewriting operation using a single-pushout (SPO) technique [13]. The aim is to develop a natural, stable foundation for Kappa, and the use of morphisms

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 276–288

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

illuminates choices made in earlier work. The foundation will provide a basis for connections to existing work (such as [3, 9]) on graph rewriting. Critically, the semantics captures side-effects in the application of rules; these are important in the efficient and natural representation of complex systems, so the careful analysis of them underpins the provision of a rewriting semantics for the whole of Kappa. This is a significant progression from the rewriting semantics presented in [7], where, with the aim of developing a theory of dynamic restriction of reductions by type, it was appropriate to study a side-effect free form of Kappa, specifically one without external links and where only whole connected components could be deleted, that permitted a double-pushout (DPO) semantics. As we shall see, DPO rewriting could not have been used without restricting to a side-effect-free fragment of Kappa.

In the second half of the paper, we introduce forms of *trajectory compression*, novel causality analyses that have been implemented in KaSim, the Kappa simulator [14]. When a specified pattern is seen in the state during simulation, for example two of a particular kind of agent being linked together, they allow the automatic generation of a refined causal account of the rule applications that led to the production of the pattern, called by biologists its *pathway*. Traditional techniques fail to provide sufficiently concise histories, leaving in place rule applications that a biologist would not expect to see in a pathway; compression addresses this important problem. The forms of compression introduced progressively increase in their ability to remove actions that are irrelevant to the production of the pattern, for example being able to remove pairs of events that remove and subsequently re-introduce structure upon which the pattern depends. It should be emphasized that, though we present compression for Kappa, we see these as *general* operations. When a user is involved in verifying any kind of model based on rewriting and the model indicates that a specified kind of event of interest can occur, there is the potential to apply these forms of compression to provide the modeller with a concise diagnosis for the event's occurrence.

2 Kappa

Kappa models how structures called *mixtures* are affected by *rules*. A mixture describes the state of *sites* on entities called *agents*. Agents and sites are labelled; each agent has at most one site with each label, and a *signature* governs which sites are allowed on each type of agent. Sites can be *linked* to other sites and can also be tagged to record any other *internal properties* that hold. A natural level of abstraction for viewing biochemical pathways is to view proteins as agents, the signature lists the (functional) sites of each type of protein, and internal properties can be used for example to model the phosphorylation state of each site.

Rules are applied to transform mixtures. Each rule consists of three *site graphs* (patterns): a left-hand side, a domain of definition and a right-hand side. A rule can be applied when its left-hand side is *matched* in the mixture. The domain of definition is a sub-pattern of the left-hand side; anything matched by something in the left-hand outside the domain of definition is deleted by application of the rule. The right-hand side extends the domain of definition with new agents, links and properties to be added to the mixture.

Significantly, to reduce the number of rules that need to be written, the test pattern of a rule does does not need to specify the full state of all agents involved, but only a partial aspect of some agents; this is neatly summed-up as: *"Don't care? Don't write."* If whether the action can occur is not determined by the state of a site, the site does not have to occur in the test pattern. However, the application of a rule induces a reaction, *i.e.* a transition in which agents are fully specified, in the mixture.

Example rules and a sequence of reactions, called a *trajectory*, are presented in Figure 1.

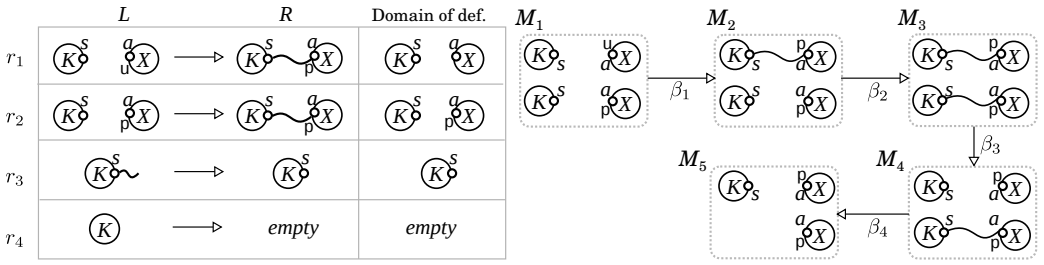
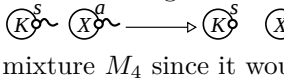


Figure 1 Example set of rules (left) and trajectory (right). Reaction β_1 is obtained by applying r_1 to the top two agents; β_2 by applying r_2 to the bottom two agents; β_3 by applying r_3 to the top-left agent; β_4 by applying r_4 to the lower-left agent.

We adopt the convention of drawing agents as large circles (labelled K and P), sites as small circles (labelled a and s) and using a sans-serif font to indicate the internal properties that hold at sites (u for unphosphorylated and p for phosphorylated).

Since rules do not have to fully specify the state of all agents involved, rule applications can have *side-effects*: changes to the mixture outside the image of the test pattern. For example, the rule r_4 in Figure 1 deletes a K-agent without reference to any of its sites; as seen in the reaction labelled β_4 in the trajectory, its application has the side-effect of removing the link to the lower X-agent. As mentioned in the introduction, Kappa allows such actions to allow the efficient representation of systems, and this will motivate our use of SPO rewriting to capture these highly intricate operations. This is in contrast to rewriting as described in [7], which considered a side-effect free fragment of Kappa, in which, for example, any deleted agent has to have all its sites specified in the rule.

To allow further compact representation, test patterns do not need to specify both ends of links: links can be one-ended, signifying the existence of a link to *some* agent; we call such links *external* links. Rule r_3 which allows reaction β_3 shows such a link being used to specify a link to be deleted. External links in a pattern can additionally specify the type of agent to which the link must connect and, optionally, the identifier of the site. In this paper, for conceptual clarity, we shall take the view that the right-hand side of an applied rule should be matched in the produced mixture. Some care will therefore be needed to manage the interaction of side-effects and external links; we shall do so by requiring the images of external links under matchings to connect to agents outside the image of the matching. Thereby, for example, we rule out application of the rule  (with domain of definition equal to the right-hand side) to the mixture M_4 since it would specify that the link connecting the lower two agents were simultaneously to be destroyed and preserved. The techniques presented in this paper are, however, completely amenable to other design decisions such as choosing to favour deletion in these cases, and the more abstract framework referred-to in the conclusion is of use when studying them.

We briefly remark that, in full Kappa, rules can be equipped with *rate constants*: values that influence how likely it is that an occurrence of a rule's test pattern in the mixture leads to application of the rule. In this paper, however, it will not be necessary to consider any stochastic aspects of the calculus.

3 Σ-graphs and morphisms

Graphs with a given signature, Σ-*graphs*, shall play a central rôle in the semantics for the Kappa that we now proceed to give: they shall include *mixtures* and represent patterns (*site graphs*), and can be used to represent types (*contact graphs*).

▶ **Definition 1.** A *signature* is a 4-tuple $\Sigma = (\Sigma_{\mathsf{ag}}, \Sigma_{\mathsf{st}}, \Sigma_{\mathsf{ag-st}}, \Sigma_{\mathsf{prop}})$, where Σ_{ag} is a set of *agent types*, Σ_{st} is a set of *site identifiers*, $\Sigma_{\mathsf{ag-st}} : \Sigma_{\mathsf{ag}} \to \mathcal{P}_{\mathsf{fin}}(\Sigma_{\mathsf{st}})$ is a *site map*, and Σ_{prop} is a set of *internal property identifiers*.

The set of agent types Σ_{ag} consists of labels to describe the nature of the agents of interest, for example the set of *kinds* proteins to be considered, and we use capital letters $A, B, A', \ldots$ to range over them. The set Σ_{st} represents the set of labels that can appear to identify sites on agents and is ranged over by $i, j, i', \ldots$. The function $\Sigma_{\mathsf{ag-st}}$ specifies the site identifiers that can be present on agents: any site on an agent of type A must be labelled with an identifier in $\Sigma_{\mathsf{ag-st}}(A)$. Finally, the set Σ_{prop} indicates the set of internal properties that sites might possess; for example, $\{\mathsf{u}, \mathsf{p}\}$ to represent 'unphosphorylated' and 'phosphorylated'.

As discussed in the previous section, there are three forms of external link. The first form, written $-$, indicates just that the link connects to some other site. The second form, written A for some $A \in \Sigma_{\mathsf{ag}}$, indicates that the link connects to some site on some agent of type A. The final form, written (A, i) for some $A \in \Sigma_{\mathsf{ag}}$ and $i \in \Sigma_{\mathsf{ag-st}}(A)$, indicates that the link connects to site i on some agent of type A. The set of all possible external link labels is defined to be $\mathsf{Ext} = \{-\} \cup \Sigma_{\mathsf{ag}} \cup \{(A, i) \ : \ A \in \Sigma_{\mathsf{ag}} \ \& \ i \in \Sigma_{\mathsf{ag-st}}(A)\}$.

▶ **Definition 2.** A Σ-graph comprises a set $\mathcal{A}$ of *agents*, an *agent type* assignment $\mathsf{type} : \mathcal{A} \to \Sigma_{\mathsf{ag}}$, a set $\mathcal{S}$ of *sites* satisfying $\mathcal{S} \subseteq \{(n, i) \ : \ n \in \mathcal{A} \ \& \ i \in \Sigma_{\mathsf{ag-st}}(\mathsf{type}(n))\}$, a symmetric *link* relation $\mathcal{L} \subseteq (\mathcal{S} \cup \mathsf{Ext})^2 \setminus \mathsf{Ext}^2$, and a *property* set p_k for each $k \in \Sigma_{\mathsf{prop}}$ satisfying

$$p_k \subseteq \{(n, i) \ : \ n \in \mathcal{A} \ \& \ i \in \Sigma_{\mathsf{ag-st}}(\mathsf{type}(n))\}$$

Note that any agent has at most one site with any given identifier: an agent n cannot have two sites identified i. In patterns, we will represent the absence of any link on a site through the site being in $\mathcal{S}$ but not occurring in any element of the link relation. Finally, the set p_k represents the set of sites that have internal property k. It is not in general the case that $p_k \subseteq \mathcal{S}$. The intuition is that we wish the set $\mathcal{S}$ to be the set of sites for which we represent knowledge of *linkage*. In a mixture, this will be all sites, but in a pattern it will be the sites that we require either to be or not to be linked. If we do not care about a site's link state in a pattern, it will not be in $\mathcal{S}$, but we may still wish to modify or test its internal properties, so it may be in the set p_k for some k.

It is useful to introduce a few notational conventions. For a Σ-graph G, we write $\mathcal{A}_G$ for its set of agents, type_G for its typing function, $\mathcal{S}_G$ for its set of link sites, $\mathcal{L}_G$ for its link relation and $p_{k,G}$ for the set of sites satisfying property k. We use m, n to range over agents and x, y to range over elements of $\mathcal{S} \cup \mathsf{Ext}$.

Mixtures, representing the state to which rules are applied, and *site graphs*, representing patterns, are forms of Σ-graph. In particular, site graphs have no links that immediately loop back to the same site and have at most one link from any site. Mixtures additionally specify the link state of all sites and have no external links.

▶ **Definition 3.** A *site graph* is a Σ-graph such that its link relation $\mathcal{L}$ is *irreflexive* and if $((n, i), x) \in \mathcal{L}$ and $((n, i), y) \in \mathcal{L}$ then $x = y$, for all n, i, x and y. A *mixture* is a site graph that additionally satisfies $\mathcal{L} \subseteq \mathcal{S} \times \mathcal{S}$ and $\mathcal{S} = \{(n, i) \ : \ n \in \mathcal{A} \ \& \ i \in \Sigma_{\mathsf{ag-st}}(\mathsf{type}(n))\}$.

It should be noted that Kappa as presented here is slightly different from past presentations in that we allow a *set* of properties to hold at a site in a mixture instead of precisely one. This streamlines the coming account of rewriting, makes no difference to expressivity, and the old definition could be handled with a few straightforward minor changes.

3.1 Homomorphisms and partial morphisms

Homomorphisms between Σ-graphs are structure-preserving functions from the agents of one Σ-graph to the agents of another. They preserve structure in the sense of preserving the presence of sites, preserving properties held on sites and ensuring that if there is a link on a site, the corresponding site on the image of the agent has a link higher in the *link information order*. Given a typing function type, this is the least reflexive, transitive relation $\leq_{\mathsf{type}}$ s.t. for all $A \in \Sigma_{\mathsf{ag}}$ and $i \in \Sigma_{\mathsf{ag-st}}(A)$ and n s.t. $\mathsf{type}_G(n) = A$: $\quad - \leq_{\mathsf{type}} A \leq_{\mathsf{type}} (A, i) \leq_{\mathsf{type}} (n, i)$

▶ **Definition 4.** A *homomorphism* of Σ-graphs $h : G \to H$ is a (total) function on agents $h : \mathcal{A}_G \to \mathcal{A}_H$ satisfying:

- $\mathsf{type}_G(n) = \mathsf{type}_H(h(n))$ for all $n \in \mathcal{A}_G$
- if $(n, i) \in \mathcal{S}_G$ then $(h(n), i) \in \mathcal{S}_H$
- $\{(h(n), i) \ : \ (n, i) \in p_{k,G}\} \subseteq p_{k,H}$ for all $k \in \Sigma_{\mathsf{prop}}$
- if $((n, i), x) \in \mathcal{L}_G$ then there exists y s.t. $((h(n), i), y) \in \mathcal{L}_H$ and $\hat{h}(x) \leq_{\mathsf{type}_H} y$, where $\hat{h}(m, j) = (h(m), j)$ for any $(m, j) \in \mathcal{S}_G$ and $\hat{h}(x) = x$ for any $x \in \mathsf{Ext}$.

It is worth remarking that we could have taken homomorphisms to be functions on agents, sites, links and properties as in [7]. Since in this paper we will primarily focus on site graphs, no difference arises from using this simpler definition.

▶ **Definition 5.** A partial morphism $h : G \rightharpoonup H$ between site graphs G and H is a span $G \hookleftarrow D \xrightarrow{h} H$ where h is a homomorphism and D is a site graph that is a subgraph of G, *i.e.*: $\mathcal{A}_D \subseteq \mathcal{A}_G$ and $\mathcal{S}_D \subseteq \mathcal{S}_G$ and $\mathcal{L}_D \subseteq \mathcal{L}_G$ and $p_{k,D} \subseteq p_{k,G}$ for all $k \in \Sigma_{\mathsf{prop}}$.

We will write $\mathrm{dom}(h)$ for the site graph representing the domain of definition of a partial morphism h, and allow ourselves to write $n \in \mathrm{dom}(h)$ to mean that n is in its set of agents, $(n, i) \in \mathrm{dom}(h)$ to mean that (n, i) is in its set of link sites, and so on. We additionally write $(n, i) \in \mathrm{dom}_{\mathsf{prop},k}(h)$ if (n, i) is in its set representing the property k.

Partial morphisms $f : G \rightharpoonup H$ and $g : H \rightharpoonup K$ compose in the usual way, with the domain of definition of their composition $\mathrm{dom}(g \circ f)$ containing elements (agents/sites/links) of $\mathrm{dom}(f)$ such that their image under f is in $\mathrm{dom}(g)$. This corresponds to taking a pullback of the homomorphism $f_0 : \mathrm{dom}(f) \to H$ against $\mathrm{dom}(g) \hookrightarrow H$ in the category Σ-**Site** of site graphs with homomorphisms between them. We write Σ-$\mathbf{Site}_*$ for the category of site graphs connected by partial morphisms.

4 Rewriting

Matchings will be used to express how patterns (site graphs) are found in mixtures. There are three key aspects. Firstly, distinct agents in patterns must match distinct agents in the mixture. Secondly, sites with no link in the pattern should have no link in the mixture, allowing patterns to express the absence of links on sites. Finally, as discussed in Section 2, the image of an external link in the pattern cannot connect to any agent in the image of the pattern. We write $e : G \rightarrowtail H$ to signify that e is a matching.

▶ **Definition 6.** A *matching* of the site graph G in H is an injective homomorphism $e : G \to H$ such that, for all $(n, i) \in \mathcal{S}_G$:

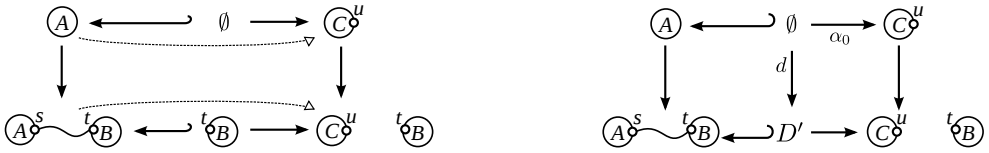


Figure 2 An SPO application (left) that cannot be a DPO application (right)

- if there exists y s.t. $((e(n),i),y) \in \mathcal{L}_H$ then there exists x such that $((n,i),x) \in \mathcal{L}_G$, and
- if there exist m and j s.t. $((e(n),i),(e(m),j)) \in \mathcal{L}_H$ then $((n,i),(m,j)) \in \mathcal{L}_G$.

Every rule is associated with a single *action map*, a special kind of partial morphism, to describe its effect. Via matchings, action maps specify what is deleted from the mixture when they are applied and what is added: anything matched by the left-hand side but not in the domain of definition is deleted by application of the rule, and anything added to the domain of definition in the progression to the right hand side is created.

Action maps have three important aspects. Firstly, the right leg of the span is injective, so rules cannot 'merge' agents together. Secondly, we wish it always to be the case that if the left-hand side of a rule matches part of the mixture, following application of the rule to that part, it matches the right-hand side of the rule. Finally, we wish to ensure that action maps preserve the property of being a mixture. Property (2) below ensures that sites are not added to or deleted from existing agents: we might otherwise introduce a site onto a preserved agent that failed to match due to the presence/absence of links, or fail to generate a mixture by deleting a site from a preserved agent. Property (3) ensures that any preserved external link is not promoted up the link information order; this is again to ensure that the right-hand side of the pattern will be consistent with the obtained mixture. Property (4) ensures that any link that is created is not an external link. Property (5) ensures that if an agent is created then so are all the sites consistent with the signature.

▶ **Definition 7.** An *action map* $\alpha : L \rightharpoonup R$ is a partial morphism of site graphs s.t.

1. α is partial injective, i.e. for all $m,n \in \mathrm{dom}(\alpha)$, if $\alpha(n) = \alpha(m)$ then $n = m$
2. for any $n \in \mathrm{dom}(\alpha)$, $\quad (n,i) \in \mathcal{S}_L$ iff $(\alpha(n),i) \in \mathcal{S}_R$ iff $(n,i) \in \mathrm{dom}(\alpha)$
3. for any $x \in \mathsf{Ext}$ and $((m,i),x) \in \mathrm{dom}(\alpha)$, $\quad ((\alpha(m),i),x) \in \mathcal{L}_R$
4. if $((m,i),x) \in \mathcal{L}_R$ and $\nexists n,y$ s.t. $\alpha(n) = m$ and $((n,i),y) \in \mathrm{dom}(\alpha)$ then there exists $(p,j) \in \mathcal{S}_R$ s.t. $x = (p,j)$.
5. if $m \in \mathcal{A}_R$ and $m \notin \mathrm{image}(\alpha)$ then $(m,i) \in \mathcal{S}_R$ for all $i \in \Sigma_{\mathsf{ag-st}}(\mathsf{type}(m))$.

The effect of applying a rule to a mixture is characterized abstractly as the following pushout in Σ-**Site**$_*$; this result will be used extensively in the following section on compression and causality. The result is dependent on the definitions of matchings and action maps since, in general, Σ-**Site**$_*$ does not have pushouts.

▶ **Theorem 8.** *Given an action $\alpha : L \rightharpoonup R$ and a matching $e : L \rightarrowtail M$, there is a pushout in the category Σ-**Site**$_*$ as follows. Furthermore, N is a mixture, u is a matching and β is an action map.*

$$\begin{array}{ccc} L & \xrightarrow{\alpha} & R \\ {\scriptstyle e}\downarrow & & \downarrow{\scriptstyle u} \\ M & \xrightarrow{\beta} & N \end{array}$$

We now give a simple example showing the necessity, due to side-effects, of SPO rather than DPO rewriting. The SPO characterization gives the application to the left of Figure 2

where the deletion of an A-agent has the side-effect of deleting a link. DPO rewriting allows a rule with an action map $\alpha : L \rightarrowtriangle R$ with domain D and right leg $\alpha_0 : D \to R$ to be applied to M to yield M' if there is a morphism $d : D \to D'$ such that M is the pushout of $D \hookrightarrow L$ against d and M' is the pushout of $\alpha_0 : D \to R$ against d. Hence, for DPO rewriting to produce the same derivation, there would have to exist D' that would allow both the squares in the diagram to the right of Figure 2 to be pushouts, which is clearly impossible.

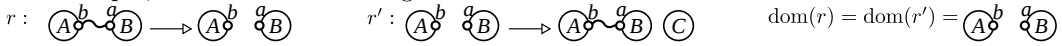
5 Trajectory compression

The pushout theorem described in the previous provides a foundation for understanding how KaSim, the Kappa simulator, simulates biochemical systems. Given an initial mixture and a set of rules described by action maps, KaSim rewrites mixtures to obtain a *trajectory*, a finite sequence of mixtures connected by action maps. Each step in the sequence is justified by a pushout in which each action map α_i is associated with a rule r_i:

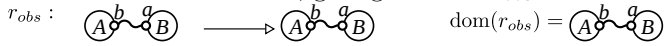
$$M_1 \xrightarrow{\beta_1} M_2 \cdots\cdots M_n \xrightarrow{\beta_n} M_{n+1} \quad \text{where} \quad \begin{array}{ccc} L_i & \xrightarrow{\alpha_i} & R_i \\ {\scriptstyle e_i}\downarrow & & \downarrow{\scriptstyle u_i} \\ M_i & \xrightarrow{\beta_i} & M_{i+1} \end{array}$$

One of the key practical uses of a trajectory is to form a causal account called a *pathway* of how specified patterns (site graphs) called *observables* come to exist in the mixture. The occurrence of an observable can be regarded simply as the application of a rule that tests if a given pattern is matched, making no change to the mixture. A natural first approach is therefore based on consideration of the independence of consecutive actions, called sequential independence in graph rewriting, the intention being to draw upon the well-developed theory of independence models for concurrency. Approximately, two consecutive rule applications are independent if they overlap only on their preserved parts. Critically, any consecutive pair of independent rule applications can be permuted, applying the same rules to the same agents, to yield another valid trajectory to the same ultimate state. Using independence, we may derive from the trajectory a partial order of causal dependence: a rule application causally depends on the rule applications that it always occurs after, under any sequence of permutations of consecutive independent rule applications. This construction forms part of the adjunction between event structures and Mazurkiewicz trace languages in [15] . The pathway of a given observable in a trajectory is the restriction of this partial order to the rule applications upon which the rule application representing the observable causally depends. In general, since the pathway is constructed by removing unnecessary rule applications from the trajectory, we call it the *Mazurkiewicz compression* of the original trace.

In practice, however, this classical approach produces pathways that contain rule applications that a biologist would normally ignore, leaving in place sequences of rule applications that have no combined effect from the perspective of the ultimate production of the observable. For example, consider the following two rules:



Application of r to a mixture exactly the same as its left-hand side followed by application of r' first destroys the link and then re-creates it, in the process creating a C agent. Let the observable be that sites a and b are linked, giving the rule r_{obs}:



There is a trajectory consisting of r, then r' and then finally r_{obs}. Considering consecutive rule applications, the observable cannot occur immediately prior to the application of r', and r' cannot be applied until r has applied, so no Mazurkiewicz compression of this trajectory

with respect to the observable can take place. Mazurkiewicz compression therefore cannot compress to show that the observable is matched by the same part of the graph in the initial mixture. In general, it leaves in place sequences of events that needlessly cycle in removing and adding structure tested by the observable.

The weakness of Mazurkiewicz compression is addressed by *weak compression*, a novel technique that draws its power from the ability simply to test if rules can be applied to the same agents at *arbitrary* prior points in the given trajectory whilst ensuring that the observable is eventually matched in the same way. This is in contrast to the consideration of independence of rules only with their *immediately* prior rule application. Though in this paper we only discuss its application to Kappa, there is potentially great scope for applying it in other rewriting settings such as traditional graph rewriting and multiset rewriting.

Finally, we introduce *strong* compression, which lifts the requirement above that the rule generating the action being compressed is applied to same agents. Though this seemingly loses a significant amount of fidelity in trajectories, in many situations the particular identity of the agents to which rules are applied is not of significance. For example, a kinase may be required to be linked to a protein for a particular sequence of phosphorylation rules to be applied (this is called *processive phosphorylation*); we often wish to disregard the possibility of the link being broken mid-sequence and then some other instance of the same kind of kinase re-establishing the link for the process to continue. It is also worth noting that strongly-compressed traces of a given observable can be extracted *statically*, *i.e.* without the need to run any simulation.

Though in this paper we focus on their foundations, both weak and strong compression have been implemented using constraint satisfaction techniques, weak in KaSim and both in the earlier *simplx* simulator, and they have been found to be of real value when analysing realistic models. The syntax to invoke them and further examples are available at [14].

5.1 Weak compression

Define the equivalence $\sim_{\mathsf{ag}}$ to relate partial morphisms $f, g : G \to H$ iff $\mathcal{A}_{\mathrm{dom}(f)} = \mathcal{A}_{\mathrm{dom}(g)}$ and $\forall n \in \mathcal{A}_{\mathrm{dom}(f)} : f(n) = g(n)$. Partial morphisms are $\sim_{\mathsf{ag}}$-related iff they are between the same site graphs and are equally defined on agents, though not necessarily on sites, properties or links.

Suppose that we have an action map $\beta : M \twoheadrightarrow M_0$ and there is a rule r_0 with action map $\alpha_0 : L_0 \twoheadrightarrow R_0$ and a matching $e_0 : L_0 \rightarrowtail M_0$. The application of r_0 to M_0 can be *weakly compressed* to occur prior to the action β, which may represent the *composition* of action maps induced by rule applications, if there is a matching $e'_0 : L_0 \rightarrowtail M$ such that $\beta \circ e'_0 \sim_{\mathsf{ag}} e_0$. This implies that r_0 can be applied to the same agents (tracking their identity through β) in M as it was in M_0 through the original matching.

▶ **Lemma 9.** *Let $\beta : M \twoheadrightarrow M_0$ and $\alpha_0 : L_0 \twoheadrightarrow R_0$ and there be a matching $e_0 : L_0 \rightarrowtail M_0$. Let M_1 be a pushout of α_0 against e_0 with pushout morphisms $\beta_0 : M_0 \twoheadrightarrow M_1$ and $u_0 : R_0 \rightarrowtail M_1$. If there is a matching $e'_0 : L_0 \rightarrowtail M$ such that $\beta \circ e'_0 \sim_{\mathsf{ag}} e_0$ then, letting M'_0 be the pushout of α_0 against e'_0 with pushout morphisms $\beta'_0 : M \twoheadrightarrow M'_0$ and $u'_0 : R_0 \rightarrowtail M'_0$, there is a morphism $\gamma_0 : M'_0 \to M_1$, unique up to $\sim_{\mathsf{ag}}$, such that $\gamma_0 \circ \beta'_0 \sim_{\mathsf{ag}} \beta_0 \circ \beta$ and $\gamma_0 \circ u'_0 \sim_{\mathsf{ag}} u_0$. Furthermore, γ_0 is an action map.*

The situation can be summarized as in the left of Figure 3, drawing the pushouts M'_0 and M_1 described in Theorem 8. Note that the map γ_0 might not be derivable as a trajectory of rules in the given Kappa system. It describes the *residual* of the compression: an action

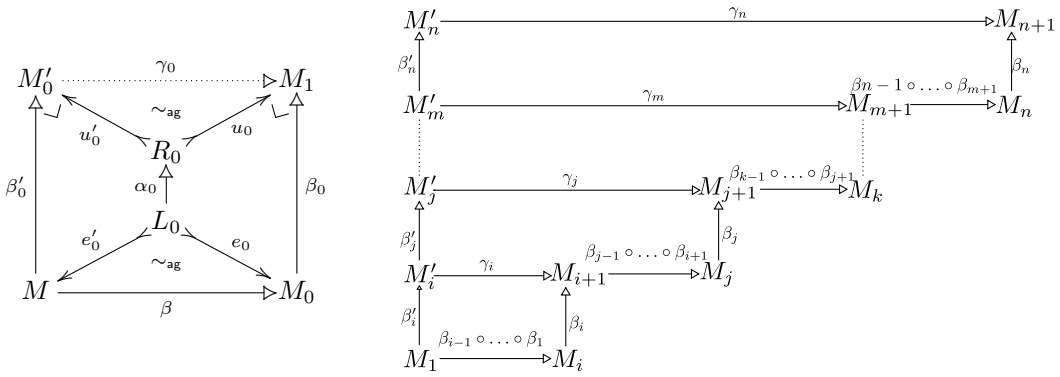


Figure 3 Steps of compression. Left: A single step of compression. Right: Multiple steps of compression, each justified by a single step of compression as drawn on the left.

summarizing the effect of the intermediate actions on the mixture formed by moving the application of the action map $\alpha_0 : L_0 \rightarrowtail R_0$ forward, excluding the modified part.

If $\beta \circ e_0' = e_0$ as opposed to $\beta \circ e_0' \sim_{\mathsf{ag}} e_0$, we call the compression *simple*. All permutations of independent individual actions will be simple, though the discussion following the example at the end of this section will explain why weak compression is more often of interest. In the case of simple compressions, $\sim_{\mathsf{ag}}$ in Lemma 9 can be replaced by equality, and the partial morphism γ_0 exists as a consequence of the left-hand pushout.

Suppose that we are given a trajectory $M_1 \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_{n-1}} M_n \xrightarrow{\beta_n} M_{n+1}$ derived as shown at the beginning of Section 5 and that we wish to produce a causal account for the last action, $\beta_n : M_n \to M_{n+1}$. Intuitively, the goal is to remove as many rule applications as possible from this original trajectory, leaving in place β_n, to yield a valid trajectory from M_1 where each of the rule applications that are selected to remain involve matching the same agents as in the original trajectory. The standard notions of independence on the compressed trajectory can then be applied to obtain a pathway.

Formally, a set of indices $\{i, j, k, \ldots, m\} \subseteq \{1, \ldots, n-1\}$, taking $i < j < k < \ldots < m$, represents a weak compression if there is a sequence of compression steps as drawn in the right of Figure 3. The first step is generated by a matching $e_i' : L_i \to M_1$ commuting (on agents) with the original matching through the composition of the prior rule applications, *i.e.* $e_i \sim_{\mathsf{ag}} \beta_{i-1} \circ \ldots \circ \beta_1 \circ e_i'$. This generates a rule application $\beta_i' : M_1 \to M_i'$ and a residual $\gamma_i : M_i' \to M_{i+1}$. Each later step requires there to exist a matching that commutes (on agents) through the composition of the prior rule applications and the previous residual. For example, for j, there must exist a matching $e_j' : L_j \to M_i'$ such that $\beta_{j-1} \circ \ldots \circ \beta_{i+1} \circ \gamma_i \circ e_j' \sim_{\mathsf{ag}} e_j$; this generates a rule application $\beta_j' : M_i' \to M_j'$ and a residual $\gamma_j : M_j' \to M_{j+1}$. Finally, we require a single step of compression to establish that there is an application of the same rule as that generating β_n in M_m', with the matching being unchanged on agents.

If $\{i, j, k, \ldots, m\}$ is a minimal (with respect to set inclusion) representation of a weak compression, the trajectory $M_1 \overset{\beta_i'}{\rightarrowtail} M_i' \cdots \overset{\beta_m'}{\rightarrowtail} M_m' \overset{\beta_n'}{\rightarrowtail} M_n'$ is *maximally weak compressed* with respect to the rule application $M_n \overset{\beta_n}{\rightarrowtail} M_{n+1}$.

▶ **Example 10.** Consider the following site graph G_0:

This represents an observable matched when there are two X-type agents with no link at their a sites and, on at least one, a is phosphorylated.

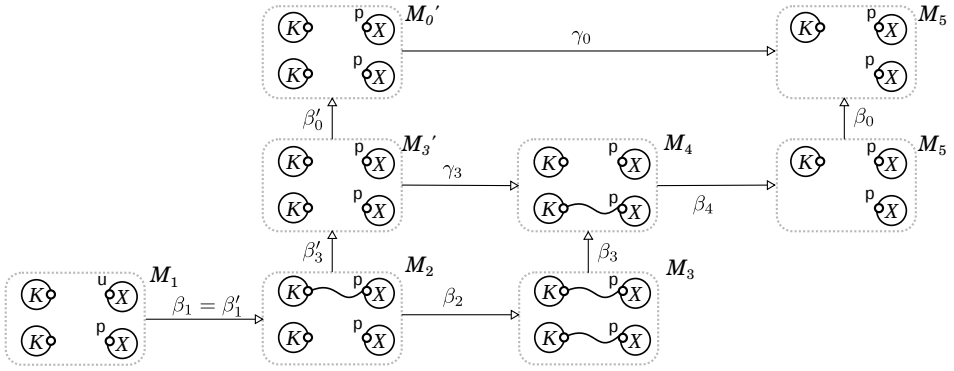


Figure 4 An example of weak compression. Sites on X are a and on K are s.

Returning to the Kappa rules and trajectory in Figure 1, this matches the mixture M_5. Let the matching $e_0 : G_0 \rightarrowtail M_5$ take the left X in G_0 to the top X in M_5 and the right X in G_0 to the bottom X in M_5. Weak compression can be applied as shown in Figure 4 to compress the trajectory $M_1 \xrightarrow{\beta_1} M_2 \xrightarrow{\beta_2} M_3 \xrightarrow{\beta_3} M_4 \xrightarrow{\beta_4} M_5 \xrightarrow{\beta_0} M_5$ where β_0 is the action testing the observable G_0. The residual map γ_3 records the creation of a link between the bottom two agents. The domain of definition of γ_0 is equal to M_4 with the lower K-agent removed, but otherwise γ_0 acts as the identity on agents. The result is a maximally compressed trajectory with the sequence of rule applications β_1', β_3' and β_0', showing that the pattern can be matched in the same way as it was for the original trajectory simply following application of the rule r_1 to phosphorylate the site a on the top X, and r_3 to remove the created link. We have 'compressed' the trajectory by removing the actions that unnecessarily affected the bottom X-agent. Mazurkiewicz compression could not have been applied to this trajectory since the action that tests the observable is not independent of the rule application that removes the link from the lower X-agent, which itself is not independent of the rule application that creates this link.

In fact, the example above shows a simple compression. Whilst, as we have seen, simple compression allows a rule application to be pushed back along a trajectory that temporarily adds and then removes links or properties that prevent application of the rule, it does not allow the event to be pushed back if links or properties are temporarily *removed*. This asymmetry is addressed by requiring commutation up to $\sim_{\mathsf{ag}}$ in *weak* compression.

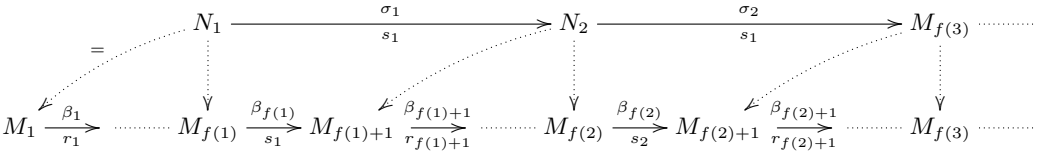
We conclude this section by noting that, in contrast with Mazurkiewicz compression, there are trajectories that have more than one maximal weak compression. For example, recalling the rules in Figure 1, from an initial mixture where an X-agent is linked to a K-agent, r_3, r_2 and r_4 could be applied in sequence followed by an observable testing that the site a on X is unlinked. Both the application of r_3 followed by the observable and the application of r_4 followed by the observable are maximal weak compressions.

5.2 Strong compression

Weak compression tests whether rules can be applied to act on the same agents at earlier points in the given trajectory. Strong compression relaxes this requirement: it asks whether a given rule in a trajectory can be applied through *any* matching; the rule does not have to be applied to the same agents. Though seemingly a very strong notion, as mentioned at the beginning of Section 5, it is appropriate in situations where we wish to ignore the identity of agents, for example in pathways for processive phosphorylation.

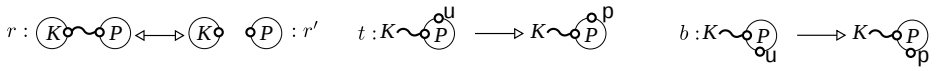
We begin by saying that a sequence of rules $s_1 \dots s_m$ is *realisable* from mixture N_1 if there is a trajectory $N_1 \overset{\sigma_1}{\to} N_2 \overset{\sigma_2}{\to} \dots \overset{\sigma_m}{\to} N_{m+1}$ and the rule applied to generate σ_i is s_i.

Consider a trajectory $M_1 \overset{\beta_1}{\to} M_2 \overset{\beta_2}{\to} \dots \overset{\beta_n}{\to} M_{n+1}$ where the rule applied to generate β_i is r_i. A *strong compression* with respect to the rule application $\beta_n : M_n \to M_{n+1}$ is a sequence $s_1 \dots s_m$ that is realisable from M_1 for which there is a monotone injection $f : \{i \ : \ 0 < i \leq m\} \to \{j \ : \ 0 < j \leq n\}$ such that $s_i = r_{f(i)}$ and $f(m) = n$. The compression is *maximal* if no rule can be removed from the sequence to yield a shorter strong compression. Labelling the action maps with the rules that generate them, we obtain:

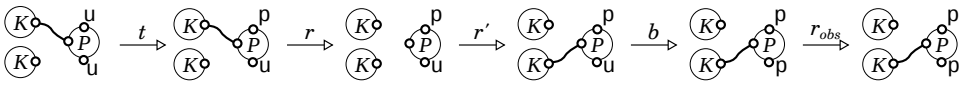


Any simple or weak compression is a strong compression, in which case the earlier constructions yield action maps for the dotted arrows. In the case of simple compression, the dotted areas commute up to equality, whereas they commute up to $\sim_{\mathsf{ag}}$ for weak compression

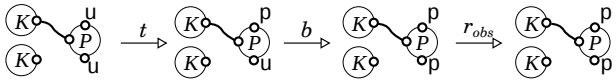
▶ **Example 11.** The following example shows how strong compression can be applied to disregard the identity of kinase agents in processive phosphorylation. In the following three rules, for clarity we have omitted site identifiers. The first is a reversible rule, so it can be applied in either direction, representing a kinase binding to/unbinding from a protein. The second and third rules represent the phosphorylation of upper and lower sites on the protein.



Taking the observable to be a P-type agent with upper and lower sites phosphorylated, we have the following trajectory (labelling reactions only with the rules that give rise to them).



Strong compression can be applied to this to obtain a trajectory consisting of rules t and b without the intervening r' and r events, where now only one of the two kinases acts.



6 Related work and conclusion

In this paper, we have seen how a semantics based on SPO graph rewriting [13] can be given to Kappa through the careful specification of various kinds of morphism. We then studied forms of trajectory compression, novel techniques that used the SPO characterization of rewriting and underlie the automatic reconstruction of biochemical pathways. This was motivated by traditional techniques based on the permutation of consecutive independent rule applications providing causal accounts with too much redundant information.

The necessity of an SPO instead of DPO semantics, due to Kappa allowing side-effectful actions, was shown in Section 4. It is worth remarking on the related work in [7] which developed a theory of dynamic restriction by type, where reduction is restricted to graphs over a given contact graph. There, a side-effect-free fragment of Kappa was considered, ruling out side effects by not having external links and only allowing rules to delete fully-specified connected components, and was shown to permit a DPO semantics.

The SPO result and characterization of compression provide an important concrete foundation for a number of areas of further research. Firstly, though there has not been space to present it here, the work in [7] on dynamic restriction translates to the SPO semantics. The analogue of the adjunction representing change of contact graph presented there, with the presence of external links introduced in this paper, can furthermore be used as a foundation for understanding *views*, special kinds of graph that are important in techniques based on static analysis for computing an approximation of the set of reachable connected components (complexes) [6]. We also intend to study how this adjunction can be used to describe techniques for providing a flexible degree of context-sensitivity in the abstraction of information flow that is used for reducing quantitative semantics [4]. Additionally, demonstrating the generality of the approach to rewriting, in [10] it is shown how it supports the addition of *regions* to site graphs to represent membranes. More abstractly, it would be interesting to determine whether sesqui-pushout rewriting [3] could fruitfully be applied to model Kappa, and to connect to the wider graph-rewriting literature, for example by comparing to [9]. Another potentially interesting connection that could be made is to the framework recently introduced in [11], which shows how negative application conditions in DPO rewriting can be specified using 'open maps'. Negative application conditions should allow us to understand abstractly constraints of the form seen in matchings, where we require the image of any unlinked site under a matching to itself be unlinked. Finally on rewriting, the work here suggests a more general framework in which to characterize pushouts of partial maps, where we represent how the preservation of elements such as links depends on the preservation of other elements such as their corresponding agents. This can be used to tease-apart the concrete proofs presented here and reveals the robustness of the pushout result when the 'externality' constraint on matchings is relaxed.

On compression, though we chose to formulate the technique for Kappa, we see such techniques for automatically providing more concise causal accounts of rule applications as being applicable much more widely in the verification of concurrent and distributed systems. Towards this, in future work we shall give an abstract framework to describe compression in any category of structures with distinguished forms of action and matching morphisms alongside a full description of its algorithmic techniques. This shall be used to show how it translates equally-well to other models such as Petri nets, traditional graph rewriting and other models for biochemical pathways such as BioNetGen[2], which differs from Kappa in allowing rules to have non-local tests (though, in terms of complexity, there is an inherent cost in dealing with these). The more abstract framework for compression may also provide a useful setting for a comparison with [12], which can be seen as defining when a single rule application can be permuted with a number of prior rule applications in the style of Mazurkiewicz compression. This would provide a form of compression between Mazurkiewicz and weak compression, in which the residual represents the composition of a number of rule applications. It is worth noting that this will be less powerful than weak compression since it will leave in place rule applications in asymmetric conflict with production of the observable. Finally, we are currently studying a framework that allows the description of other more sophisticated forms of compression lying between weak and strong, for example compression allowing the identity only of particular types of agent to be disregarded.

Acknowledgments: JH and GW gratefully acknowledge the support of the ERC through the Advanced Grant *ECSYM*, JF and JH the support of the ANR *AbstractCell* Chair of Excellence and JK the ANR Avenir *ICEBERG*.

References

1 J. A. Bachman and P. Sorger. New approaches to modeling complex biochemistry. *Nature Methods*, 8(2):130–131, 2011.

2 M. L. Blinov, J. Yang, J. R. Faeder, and W. S. Hlavacek. Graph theory for rule-based modeling of biochemical networks. In *Proc. CSB*, number 4230 in LNCS, 2006.

3 A. Corradini, T. Heindel, F. Hermann, and B. König. Sesqui-pushout rewriting. In *Proc. ICGT*, 2006.

4 V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Abstracting the differential semantics of rule-based models: exact and automated model reduction. In *Proc. LICS*, 2010.

5 V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable simulation of cellular signaling networks. In *Proc. APLAS*, 2007.

6 V. Danos, J. Feret, W. Fontana, and J. Krivine. Abstract interpretation of cellular signalling networks. In *Proc. VMCAI*, 2008.

7 V. Danos, R. Harmer, and G. Winskel. Constraining rule-based dynamics with types. *MSCS*, 2012.

8 V. Danos and C. Laneve. Formal molecular biology. *TCS*, 325, 2004.

9 H. Ehrig, J. Padberg, U. Prange, and A. Habel. Adhesive high-level replacement systems: A new categorical framework for graph transformation. In *Proc. ICGT '04*, volume 74 of *Fundamenta Informaticae*, 2006.

10 J. Hayman, C. Thompson-Walsh, and G. Winskel. Simple containment structures in rule-based modelling of biochemical systems. In *Proc. SASB*, 2011.

11 R. Heckel. DPO transformation with open maps. In *Proc. ICGT*, number 7562 in LNCS, 2012.

12 F. Hermann. Permutation equivalence of DPO derivations with negative application conditions based on subobject transformation systems. *ECEASST*, 16, 2008.

13 M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109, 1993.

14 `kappalanguage.org`. The kappa simulator.

15 G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic and the Foundations of Computer Science*. OUP, 1995.

On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks

Giorgio Delzanno[1], Arnaud Sangnier[2], Riccardo Traverso[1], and Gianluigi Zavattaro[4]

1 **DIBRIS, University of Genova, Italy**
2 **LIAFA, Univ Paris Diderot, Sorbonne Paris Cité, CNRS, France**
4 **University of Bologna, INRIA - FOCUS Research Team, Italy**

Abstract

We investigate the impact of dynamic topology reconfiguration on the complexity of verification problems for models of protocols with broadcast communication. We first consider reachability of a configuration with a given set of control states and show that parameterized verification is decidable with polynomial time complexity. We then move to richer queries and show how the complexity changes when considering properties with negation or cardinality constraints.

1998 ACM Subject Classification F.3 Logics and Meanings of Programs, F.1.1 Models of Computation

Keywords and phrases Broadcast Communication, Parameterized Verification, Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.289

1 Introduction

Broadcast communication is often used in networks in which individual nodes have no precise information about the underlying connection topology (e.g. ad hoc wireless networks). As shown in [4, 14, 15, 18, 19], this type of communication can naturally be specified in models of computation in which a network configuration is represented as a graph and in which individual nodes run an instance of a common protocol. A protocol typically specifies a sequence of control states in which a node can send a message (emitter role), wait for a message (receiver role), or perform an update of its internal state.

Already at this level of abstraction, verification of protocols with broadcast communication turns out to be a very difficult task. A formal account of this problem is given in [3, 4], where *parameterized control state reachability* is proved to be undecidable in an automata-based protocol model in which configurations are arbitrary graphs. The parameterized control state reachability problem consists in verifying the existence of an initial network configuration (with unknown size and topology) that may evolve into a configuration in which at least one node is in a given control state. If such a control state represents a protocol error, then this problem naturally expresses (the complement of) a safety verification task in a setting in which nodes have no information a priori about the size and connection topology of the underlying network.

In presence of non-deterministic reconfigurations of the network topology during an execution, parameterized control state reachability becomes decidable [3]. Reconfiguration models spontaneous node movement, i.e. each node can dynamically connect (resp. disconnect) to (resp. from) any other node in the network. Furthermore, it also models the dynamic addition (resp. removal) of nodes by means of connection to the network of a previously disconnected idle node (resp. the definitive disconnection of a previously connected

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 289–300

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

node). The decidability proof in [3] does not give exact complexity bounds of the problem; it simply gives a reduction to Petri net coverability, an ExpSpace-complete problem [13, 20]. The precise complexity of parameterized reachability was left as an open problem in [3].

In this paper we present a comprehensive analysis of the complexity of reachability problems for reconfigurable broadcast networks. We start by generalizing the problem by considering reachability queries defined over assertions that: (i) check the presence or absence of control states in a given configuration generated by some initial configuration, and (ii) cardinality queries that define lower and upper bounds for the number of occurrences of control states in a reachable configuration. In any case the problems require, at least in principle, the exploration of an infinite-state space. Indeed they are formulated for arbitrary initial configurations, and upper bounds to the number of processes per control state are not mandatory in case (ii). We then move to the analysis of the complexity of the considered problems by showing that reachability queries for constraints that only check for the presence of a control state can be checked in polynomial time. When considering both constraints for checking presence and absence of control states the problem turns out to be NP-complete. Finally, we show that the problem becomes PSpace-complete for cardinality queries.

Related Work. As mentioned in the introduction no precise complexity bounds were given for the parameterized control state reachability problem proved decidable in [3] via a reduction to Petri net marking coverability. In the present paper we attack this problem for different types of reachability queries. The interreducibility of control state reachability in models with dynamic reconfiguration, spontaneous mobility, and node-, message-, or link-failures has been formally studied in [5]. Based on the results in [5], the PTime-algorithm presented in the current paper can be applied not only to reconfigurable networks but also to a variety of protocols models with failures.

Symbolic backward exploration procedures for network protocols specified in graph rewriting have been presented in [11] (termination guaranteed for ring topologies) and [17] (approximations without termination guarantees). Decidability issues for broadcast communication in unstructured concurrent systems (or, equivalently, in fully connected networks) have been studied, e.g., in [8], whereas verification of unreliable communicating FIFO systems has been studied, e.g., in [1].

To our knowledge, exact algorithms (and relative complexity) for parameterized verification has not been studied in previous work on graph-based models of synchronous or asynchronous broadcast communication like [16, 18, 19, 15, 7, 9, 10, 14, 17, 11].

Notes. Sketches of the proofs are included in the body of the paper; detailed proofs are given in [6].

2 A Model for Reconfigurable Broadcast Networks

2.1 Syntax and semantics

Our model for reconfigurable broadcast networks is defined in two steps. We first define graphs used to denote network configurations and then define protocols running on each node. The label of a node denotes its current control state. Finally, we give a transition system for describing the interaction of a vicinity during the execution of the same protocol on each node.

▶ **Definition 2.1.** A Q-graph is a labeled *undirected graph* $\gamma = \langle V, E, L\rangle$, where V is a finite set of *nodes*, $E \subseteq V \times V \setminus \{\langle v, v\rangle \mid v \in V\}$ is a finite set of *edges*, and L is a labeling function from V to a set of labels Q.

We use $L(\gamma)$ to represent all the labels present in γ (i.e. the image of the function L). The nodes belonging to an edge are called the *endpoints* of the edge. For an edge $\langle u, v\rangle$ in E, we use the notation $u \sim_\gamma v$ and say that the vertices u and v are adjacent one to another in the graph γ. We omit γ, and simply write $u \sim v$, when it is made clear by the context.

▶ **Definition 2.2.** A process is a tuple $\mathcal{P} = \langle Q, \Sigma, R, Q_0\rangle$, where Q is a finite set of control states, Σ is a finite alphabet, $R \subseteq Q \times (\{!!a, ??a \mid a \in \Sigma\}) \times Q$ is the transition relation, and $Q_0 \subseteq Q$ is a set of initial control states.

The label $!!a$ [resp. $??a$] represents the capability of broadcasting [resp. receiving] a message $a \in \Sigma$. For $q \in Q$ and $a \in \Sigma$, we define the set $R_a(q) = \{q' \in Q \mid \langle q, ??a, q'\rangle \in R\}$ which contains the states that can be reached from the state q when receiving the message a. We assume that $R_a(q)$ is non empty for every a and q, i.e. nodes always react to broadcast messages. Local transitions (denoted by the special label τ) can be derived by using a special message m_τ such that $\langle q, ??m_\tau, q'\rangle \in R$ implies $q' = q$ for every $q, q' \in Q$ (i.e. receivers do not modify their local states).

Given a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0\rangle$, in the corresponding Reconfigurable Broadcast Network (RBN) a configuration is a Q-graph and an initial configuration is a Q_0-graph. We use Γ [resp. Γ_0] to denote the set of configurations [resp. initial configurations] associated to $\mathcal{P}$. Note that even if Q_0 is finite, there are infinitely many possible initial configurations (the number of Q_0-graphs). We assume that each node of the graph is a process that runs a common predefined protocol defined by a communicating automaton with a finite set Q of control states. Communication is achieved via selective broadcast, which means that a broadcasted message is received by the nodes which are adjacent to the sender. Nondeterminism in reception is modeled by means of graph reconfigurations. We next formalize this intuition.

Given a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0\rangle$, a reconfigurable broadcast network is defined by the transition system $RBN(\mathcal{P}) = \langle \Gamma, \to, \Gamma_0\rangle$ where the transition relation $\to \subseteq \Gamma \times \Gamma$ is such that: for $\gamma, \gamma' \in \Gamma$ with $\gamma = \langle V, E, L\rangle$, we have $\gamma \to \gamma'$ iff $\gamma' = \langle V, E', L'\rangle$ and one of the following conditions holds:

Broadcast $E' = E$ and $\exists v \in V$ s.t. $\langle L(v), !!a, L'(v)\rangle \in R$ and $L'(u) \in R_a(L(u))$ for every $u \sim v$, and $L(w) = L'(w)$ for any other node w.

Graph reconfiguration $E' \subseteq V \times V \setminus \{\langle v, v\rangle \mid v \in V\}$ and $L = L'$.

We use $\to^*$ to denote the reflexive and transitive closure of $\to$. RBN is an adequate formalism to abstractly represent broadcast communication with features like spontaneous mobility, node-, message- and link-failures.

2.2 Parameterized Reachability Problems

Given a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0\rangle$, a *cardinality constraint* φ over $\mathcal{P}$ is a formula which defines lower and upper bounds for the number of occurrences of each control state in a configuration. The formulae are defined by the following grammar, where $a \in \mathbb{N}$, $q \in Q$, and $b \in (\mathbb{N} \setminus \{0\}) \cup \{+\infty\}$:

$$\varphi ::= a \leq \#q < b \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi$$

We denote by CC the class of cardinality constraints, by CC$[\geq 1]$ the class in which negation is forbidden and atomic proposition have only the form $\#q \geq 1$ (there exists at least one occurrence of q), and finally by CC$[\geq 1, = 0]$ the class of cardinality constraints as in CC$[\geq 1]$ but where atoms can also be of the form $\#q = 0$. Given a configuration $\gamma = \langle V, E, L\rangle$

of $\mathcal{P}$ and $q \in Q$, we denote by $\#\gamma(q)$ the number of vertices in γ labeled by q, that is $\#\gamma(q) = |\{v \in V \mid L(v) = q\}|$. The satisfaction relation $\models$ for atomic formulas is defined as follows $\gamma \models a \leq \#q < b$ iff $a \leq \#\gamma(q) < b$. It is defined in the natural way for compound formulas.

We are now ready to state the cardinality reachability problem (CRP):

Input: A process $\mathcal{P}$ with $RBN(\mathcal{P}) = \langle \Gamma, \rightarrow, \Gamma_0 \rangle$ and a cardinality constraint φ.

Output: Yes, if $\exists \gamma_0 \in \Gamma_0$ and $\gamma_1 \in \Gamma$ s.t. $\gamma_0 \rightarrow^* \gamma_1$ and $\gamma_1 \models \varphi$; no, otherwise.

If the answer to this problem is yes, we will write $\mathcal{P} \models \Diamond\varphi$. Note that when dealing with the complexity of this problem we will suppose that the size of the input is the size of the process defined by the product of the number of states times the number of edges added to the size of the formula in which the integer values are encoded in unary.

We use the term *parameterized* to remark that the initial configuration is not fixed a priori. In fact, the only constraint that we put on the initial configuration is that the nodes have labels taken from Q_0 without any information on their number or connection links. As a special case we can define the control state reachability problems studied in [3] as the CRP for the simple constraint $\#q \geq 1$ (i.e. is there a reachable configuration in which the state q is exposed?). Similarly, we can define the target reachability problem studied in [3] as an instance of CRP in which control states that must not occur in a target configuration are constrained by formulas like $\#q = 0$.

According to our semantics, the number of nodes stays constant in each execution starting from the same initial configuration. As a consequence, when fixing the initial configuration γ_0, we obtain finitely many possible reachable configurations. Thus, checking if there exists γ_1 reachable from a given γ_0 s.t. $\gamma_1 \models \varphi$ for a constraint φ is a decidable problem. On the other hand, checking the parameterized version of the reachability problem is in general much more difficult. E.g. consider constraints of the form $\#q \geq 1$: CRP is undecidable for a semantics without non-deterministic graph reconfigurations [3]. In [3] it is also proved that CRP for the same class of constraints is decidable. However, the proposed decidability proof is based on a reduction to the problem of coverability in Petri nets which is known to be ExpSpace-complete [20, 13]. Since no lower-bound was provided, the precise complexity of CRP with simple constraints was left as an open problem that we close in this paper by showing that it is PTime-complete.

3 CRP restricted to constraints in CC[≥ 1]

In this section, we study CRP restricted to CC[≥ 1]. These constraints characterize configurations in which a given set of control states is present but they cannot express neither the absence of states nor the number of their occurrences. We first give a lower bound for this problem.

▶ **Proposition 3.1.** *CRP restricted to CC[≥ 1] is* PTime*-hard.*

Sketch of proof. The idea for the proof is based on a LogSpace-reduction from the Circuit Value Problem (CVP), which is know to be PTime-complete [12]. The protocol $\mathcal{P}$ built from the CVP instance has an initial state for each of the input variables which broadcasts its truth assignment, and another one for each gate of the input circuit. In the sub-protocol associated to individual gates, a process waits for messages representing inputs and then broadcasts messages representing outputs in such a way that CVP is satisfied iff $\mathcal{P} \models \Diamond \#ok \geq 1$, where ok is a state reached only when the last gate produces the expected output. ◀

We now show that CRP restricted to CC[≥ 1] is in PTIME. We first observe that, in order to decide if control state q can be reached, we can focus our attention on initial configurations in which the topology is fully connected (i.e. graphs in which all pairs of nodes are connected). Indeed, graph reconfigurations can be applied to non-deterministically transform a topology into any other one.

Another key observation is that if the control state q is reached once from the initial configuration γ_0, then it can be reached an arbitrary number of times by considering larger and larger initial configurations γ'_0. More specifically, the initial configuration γ'_0 is obtained by replicating several times the initial graph γ_0. The replicated parts are then connected in all possible ways (to obtain a fully connected topology). We can then use dynamic reconfiguration in order to mimic parallel executions of the original system and reach a configuration with several repeated occurrences of state q.

For what concerns constraints in CC[≥ 1] this property of CRP avoids the need of counting the occurrences of states. We just have to remember which states can be generated by repeatedly applying process rules. As a consequence, in order to define a decision

Algorithm 1 Computing the set of control states reachable in a RBN

Input: $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ a process
Output: $S \subseteq Q$ the set of reachable control states in $RBN(\mathcal{P})$
  $S := Q_0$
  $oldS := \emptyset$
  while $S \neq oldS$ **do**
    $oldS := S$
    for all $\langle q_1, !!a, q_2 \rangle \in R$ such that $q_1 \in oldS$ **do**
      $S := S \cup \{q_2\} \cup \{q' \in Q \mid \langle q, ??a, q' \rangle \in R \wedge q \in oldS\}$
    end for
  end while

procedure for checking control state reachability we can take the following assumptions: (i) forget about the topology underlying the initial configuration; (ii) forget about the number of occurrences of control states in a configuration; (iii) consider a single symbolic path in which at each step we apply all possible rules whose preconditions can be satisfied in the current set and then collect the resulting set of computed states.

We now formalize the previous observations. Let $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ be a process with $RBN(\mathcal{P}) = \langle \Gamma, \rightarrow, \Gamma_0 \rangle$ and let $\texttt{Reach}(\mathcal{P})$ be the set of reachable control states equals to $\{q \in Q \mid \exists \gamma \in \Gamma_0 . \exists \gamma' \in \Gamma \text{. s.t. } \gamma \rightarrow^* \gamma' \text{ and } q \in L(\gamma')\}$. We will now prove that Algorithm 1 computes $\texttt{Reach}(\mathcal{P})$. Let S be the result of the Algorithm 1 (note that this algorithm necessarily terminates because the **while**-loop is performed at most $|Q|$ times). We have then the following lemma.

▶ **Lemma 3.2.** *The two following properties hold:*
(i) *There exist two configurations $\gamma_0 \in \Gamma_0$ and $\gamma \in \Gamma$ such that $\gamma_0 \rightarrow^* \gamma$ and $L(\gamma) = S$.*
(ii) *$S = \texttt{Reach}(\mathcal{P})$.*

Proof. We first prove (i). We denote by $S_0, S_1, \ldots, S_n$ the content of S after each iteration of the loop of the Algorithm 1. We recall that an undirected graph $\gamma = \langle V, E, L \rangle$ is complete if $\langle v, v' \rangle \in E$ for all $v, v' \in V$. We will now consider the following statement: for all $j \in \{0, \ldots, n\}$, for all $k \in \mathbb{N}$, there exists a complete graph $\gamma_{j,k} = \langle V, E, L \rangle$ in Γ verifying the two following points:

1. $L(\gamma_{j,k}) = S_j$ and for each $q \in S_j$, the set $\{v \in V \mid L(v) = q\}$ has more than k elements (i.e. for each element q of S_j there are more than k nodes in $\gamma_{j,k}$ labeled with q),

2. there exits $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \to^* \gamma_{j,k}$.

To prove this statement we reason by induction on j. First, for $j = 0$, the property is true, because for each $k \in \mathbb{N}$, the graph $\gamma_{0,k}$ corresponds to the complete graphs where each of the initial control states appears at least k times. We now assume that the property is true for all naturals smaller than j (with $j < n$) and we will show it is true for $j + 1$. We define C_a as the set $\{\langle\langle q_1, !!a, q_2\rangle, \langle q, ??a, q'\rangle\rangle \in R \times R \mid q_1, q \in S_j\}$ and M its cardinality. Let $k \in \mathbb{N}$ and let $N = k + 2 * k * M$. We consider the graph $\gamma_{j,N}$ where each control state present in S_j appears at least N times (such a graph exists by the induction hypothesis). From $\gamma_{j,N}$, we build the graph $\gamma_{j+1,k}$ obtained by repeating k times the following operations:

- for each pair $\langle\langle q_1, !!a, q_2\rangle, \langle q, ??a, q'\rangle\rangle \in C_a$, select a node labeled by q_1 and one labeled by q and update their label respectively to q_2 and q' (this simulates a broadcast from the node labeled by q_1 received by the node labeled q in the configuration in which all the other nodes have been disconnected thanks to the reconfiguration and reconnected after). Note that the two selected nodes can communicate because the graph is complete.

By applying these rules it is then clear that $\gamma_{j,N} \to^* \gamma_{j+1,k}$ and also that $\gamma_{j+1,k}$ verifies the property 1 of the statement. Since by induction hypothesis, we have that there exists $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \to^* \gamma_{j,N}$, we also deduce that $\gamma_0 \to^* \gamma_{j+1,k}$, hence the property 2 of the statement also holds. From this we deduce that (i) is true.

To prove (ii), from (i) we have that $S \subseteq \texttt{Reach}(\mathcal{P})$ and we now prove that $\texttt{Reach}(\mathcal{P}) \subseteq S$. Let $q \in \texttt{Reach}(\mathcal{P})$. We show that $q \in S$ by induction on the minimal length of an execution path $\gamma_0 \to^* \gamma$ such that $\gamma_0 \in \Gamma_0$ and $q \in L(\gamma)$. If the length is 0 then $q \in Q_0$ hence also $q \in S$. Otherwise, let $\gamma' \to \gamma$ be the last transition of the execution. We have that there exists $q_1 \in L(\gamma')$ such that $\langle q_1, !!a, q\rangle \in R$ [or $q_1, q_2 \in L(\gamma')$ such that $\langle q_1, !!a, q_3\rangle, \langle q_2, ??a, q\rangle \in R$]. By induction hypothesis we have that $q_1 \in S$ [or $q_1, q_2 \in S$]. By construction, we can conclude that also $q \in S$. ◀

Since constraints in CC[≥ 1] check only the presence of states and do not contain negation, given a configuration γ and a constraint φ in CC[≥ 1] such that $\gamma \models \varphi$, we also have that $\gamma' \models \varphi$ for every γ' such that $L(\gamma) \subseteq L(\gamma')$. Moreover, given a process $\mathcal{P}$, by definition of $\texttt{Reach}(\mathcal{P})$ we have that $L(\gamma) \subseteq \texttt{Reach}(\mathcal{P})$ for every reachable configuration γ, and by Lemma 3.2 there exists a reachable configuration γ_f such that $L(\gamma_f) = \texttt{Reach}(\mathcal{P})$. Hence, to check $\mathcal{P} \models \Diamond\varphi$ it is sufficient to verify whether $\gamma_f \models \varphi$ for such a configuration γ_f. This can be done algorithmically as follows: once the set $\texttt{Reach}(\mathcal{P})$ is computed, check if the boolean formula obtained from φ by replacing each atomic constraint of the form $\#q \geq 1$ by *true* if $q \in \texttt{Reach}(\mathcal{P})$ and by *false* otherwise is valid. This allows us to state the following theorem.

▶ **Theorem 3.3.** *CRP restricted to CC[≥ 1] is* PTime*-complete.*

Proof. The lower bound is given by Proposition 3.1. To obtain the upper bound, it suffices to remark that the Algorithm 1 is in PTime since it requires at most $|Q|$ iterations each one requiring at most $|R|^2$ look-ups (of active broadcast/receive transitions) for computing new states to be included, and also that evaluating the validity of a boolean formula can be done in polynomial time. ◀

4 CRP restricted to constraints in CC[$\geq 1, = 0$]

We consider now decidability and complexity of CRP for constraints in CC[$\geq 1, = 0$]. This kind of queries can be used to specify that a given control state is not present in a configuration (using atomic constraints of the form $\#q = 0$).

▶ **Proposition 4.1.** *CRP for constraints in CC[$\geq 1, = 0$] is* NP*-hard.*

Sketch of proof. The proof is based on a reduction of the boolean satisfiability problem (SAT), which is known to be NP-complete. The encoding of the SAT instance for a boolean formula Φ with variables in V is based on a protocol with only local transitions from a single initial state into states that encode truth assignments in $\{v, \overline{v} \mid v \in V\}$. A CC[$\geq 1, = 0$] query is then built in order to guarantee that there are no contradicting assignments to variables. The query also ensures that the selected assignments satisfy the formula Φ, where positive literals v are replaced by $\#v \geq 1$ and negative literals $\neg v$ are replaced by $\#v = 0$. ◀

We will now give an algorithm in NP to solve CRP for constraints in CC[$\geq 1, = 0$]. As for Algorithm 1, this new algorithm works on sets of control states. The algorithm works in two main phases. In a first phase it generates an increasing sequence of sets of control states that can be reached in the considered process definition. At each step the algorithm adds the control states obtained from the application of the process rules to the current set of labels. Unlike the Algorithm 1, this new algorithm does not merge different branches, i.e. application of distinct rules may lead to different sequences of sets of control states. In a second phase the algorithm only removes control states applying again process rules in order to reach a set of control states that satisfies the given constraint.

Algorithm 2 Solving CRP for constraints in CC[$\geq 1, = 0$]

Input: $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ a process and φ a constraint over $\mathcal{P}$ in CC[$\geq 1, = 0$]
Output: Does $\mathcal{P} \models \Diamond\varphi$?
  guess $S_0, \ldots, S_m, T_1, \ldots, T_n \subseteq Q$ with $m, n \leq |Q|$
  if $S_0 \not\subseteq Q_0$ **then return false**
  for all $i \in \{0, \ldots, m-1\}$ **do**
    if $S_{i+1} \notin \texttt{postAdd}(\mathcal{P}, S_i)$ **then return false**
  end for
  $T_0 = S_m$
  for all $i \in \{0, \ldots, n-1\}$ **do**
    if $T_{i+1} \notin \texttt{postDel}(\mathcal{P}, T_i)$ **then return false**
  end for
  If T_n satisfies φ **then return true else return false**

For a process $\mathcal{P} = \langle Q, \Sigma, R, Q_0 \rangle$ and a set $S \subseteq Q$, we define the operator $\texttt{postAdd}(\mathcal{P}, S) \subseteq 2^Q$ as follows: $S' \in \texttt{postAdd}(\mathcal{P}, S)$ if and only if the two following conditions are satisfied: (i) $S \subseteq S'$ and (ii) for all $q' \in S' \setminus S$, there exists a rule $\langle q, !!a, q' \rangle \in R$ such that $q \in S$ (q' is produced by a broadcast) or there exist rules $\langle p, !!a, p' \rangle$ and $\langle q, ??a, q' \rangle \in R$ such that $q, p \in S$ and $p' \in S'$ (q' is produced by a reception). In other words, all the states in $S' \in \texttt{postAdd}(\mathcal{P}, S)$ are either in S or states obtained from the application of broadcast/reception rules to labels in S. Similarly, we define the operator $\texttt{postDel}(\mathcal{P}, S) \subseteq 2^Q$ as follows: $S' \in \texttt{postDel}(\mathcal{P}, S)$ if and only if $S' \subseteq S$ and one of the following conditions hold: either $S \setminus S' = \emptyset$ or $[S \setminus S' = \{q\}$ and there exists a rule $\langle q, !!a, q' \rangle \in R$ such that $q' \in S']$ or $[S \setminus S' = \{q\}$ and there exist two rules $\langle p, !!a, p' \rangle, \langle q, ??a, q' \rangle \in R$ such that $p, p', q' \in S'$ (q is consumed by a broadcast)] or $[S \setminus S' = \{p, q\}$ and there exist two rules $\langle p, !!a, p' \rangle, \langle q, ??a, q' \rangle \in R$ such that $p', q' \in S'$ (p and q are consumed by a broadcast)].

Finally, we say that a set $S \subseteq Q$ satisfies an atom $\#q = 0$ if $q \notin S$ and it satisfies an atom $\#q \geq 1$ if $q \in S$; satisfiability for composite boolean formulae of CC[$\geq 1, = 0$] is then defined in the natural way. We have then the following Lemma.

▶ **Lemma 4.2.** *There is an execution of Algorithm 2 which answers YES on input $\mathcal{P}$ and φ iff $\mathcal{P} \models \Diamond\varphi$.*

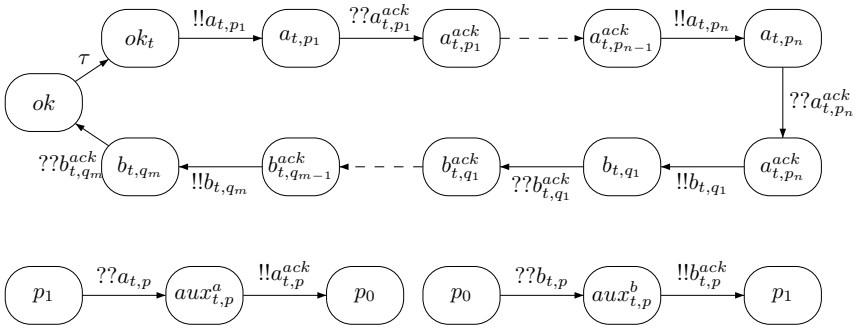


Figure 1 Simulation of a transition t with ${}^\bullet t = \{p_1, \ldots, p_n\}$ and $t^\bullet = \{q_1, \ldots, q_m\}$.

It is then clear that each check performed by the Algorithm 2 (i.e. $S_0 \subseteq Q_0$ and $S_{i+1} \in \texttt{postAdd}(\mathcal{P}, S_i)$ and $T_{i+1} \in \texttt{postAdd}(\mathcal{P}, T_i)$ and T_n satisfies φ) can be performed in polynomial time in the size of the process $\mathcal{P}$ and of the formula φ and since m and n are smaller than the number of control states in $\mathcal{P}$, we deduce the following theorem (the lower bound being given by Proposition 4.1).

▶ **Theorem 4.3.** *CRP for constraints in CC[$\geq 1, = 0$] is* NP*-complete.*

5 Complexity of CRP in Full CC

In this section we will show that CRP for the entire class of cardinality constraints CC is PSPACE-complete. First we prove the lower bound.

▶ **Proposition 5.1.** *CRP is* PSPACE*-hard.*

Proof. We use a reduction from reachability in 1-safe Petri nets. A Petri net N is a tuple $N = \langle P, T, \vec{m_0} \rangle$, where P is a finite set of places, T is a finite set of transitions t, such that ${}^\bullet t$ and $t^\bullet$ are multisets of places (pre- and post-conditions of t), and $\vec{m_0}$ is a multiset of places that indicates how many tokens are located in each place in the initial net marking. Given a marking $\vec{m}$, the firing of a transition t such that ${}^\bullet t \subseteq \vec{m}$ leads to a new marking $\vec{m'}$ obtained as $\vec{m'} = \vec{m} \setminus {}^\bullet t \cup t^\bullet$. A Petri net P is 1-safe if in every reachable marking every place has at most one token. Reachability of a specific marking $\vec{m_1}$ from the initial marking $\vec{m_0}$ is decidable for Petri nets, and PSPACE-complete for 1-safe nets [2].

Given a 1-safe net $N = \langle P, T, \vec{m_0} \rangle$ and a marking $\vec{m_1}$, we encode the reachability problem as a CRP problem for the process $\mathcal{P}$ and cardinality constraint φ defined next. For each place $p \in P$, we introduce control states p_1 and p_0 to denote the presence or absence of the token in p, respectively. Furthermore, we introduce a special control state ok. The control state is used to control the net simulation. Transitions of the controller are depicted in the upper part of Fig. 1. The first rule of the controller selects the current transition to simulate. The simulation of the transition t with ${}^\bullet t = \{p_1, \ldots, p_n\}$ and $t^\bullet = \{q_1, \ldots, q_m\}$ is defined via two sequences of messages (we denote ${}^\bullet t$ and $t^\bullet$ as sets instead of multisets because we are considering a 1-safe net and it is hence not possible that a transition consumes or produces more than one token for each place). The first one is used to remove the token from $p_1, \ldots, p_n$, whereas the second one is used to put the token in $q_1, \ldots, q_m$. To guarantee that every involved place reacts to the protocol —i.e. messages are not lost— the controller waits for an acknowledgement from each of them. Transitions of places are depicted in the lower part of Fig. 1. It is not restrictive to assume that there is only one token in the initial marking $\vec{m_0}$ (otherwise we add an auxiliary initial place and a transition that generates $\vec{m_0}$ by consuming the initial token). Let p^0 be such a place. We define the

initial states Q_0 of the process $\mathcal{P}$ as $\{p_1^0, ok\} \cup \{p_0 \mid p \in P \setminus \{p^0\}\}$, in order to initially admit control states representing the controller, the presence of the initial token, and the absence of tokens in other places. The reduction does not work if there are several copies of controller nodes and/or place representations (i.e. $p_1, p_0, \ldots$) interacting during a simulation (interferences between distinct nodes representing controllers/places may lead to incorrect results). However we can ensure that the reduction is accurate by checking the number of occurrences of states exposed in the final configuration: it is sufficient to check that only one controller and only one node per place in the net are present. Besides making this check, the cardinality constraint φ should also verify that the represented net marking coincides with $\vec{m_1}$. Namely, we define φ as follows:

$$\begin{aligned}\varphi = & \bigwedge_{p \in \vec{m_1}, t \in T} \left(\#p_1 = 1 \wedge \#p_0 = 0 \wedge \#aux_{t,p}^a = 0 \wedge \#aux_{t,p}^b = 0\right) \wedge \\ & \bigwedge_{q \notin \vec{m_1}, t \in T} \left(\#q_1 = 0 \wedge \#q_0 = 1 \wedge \#aux_{t,q}^a = 0 \wedge \#aux_{t,q}^b = 0\right) \wedge \#ok = 1 \wedge \\ & \bigwedge_{t \in T} (\#ok_t = 0) \wedge \bigwedge_{t \in T, q \in P} \left(\#a_{t,q} = 0 \wedge \#b_{t,q} = 0 \wedge \#a_{t,q}^{ack} = 0 \wedge \#b_{t,q}^{ack} = 0\right)\end{aligned}$$

Since the number of nodes stays constant during an execution, the post-condition specified by φ is propagated back to the initial configuration. Therefore, if the protocol satisfies CRP for φ, then in the initial configuration there must be one single controller node with state ok, and for each place p one single node with either state p_1 or state p_0. Under this assumption, it is easy to check that a run of the protocol corresponds precisely to a firing sequence in the 1-safe net. Thus an execution run satisfies φ if and only if the corresponding firing sequence reaches the marking $\vec{m_1}$. ◀

We now show that there exists an algorithm to solve CRP in PSpace. The main idea is to use a symbolic representation of configurations in which the behavior of a network is observed exactly for a fixed number of nodes only. For all the other nodes, we only maintain the control state they are labeled with and not their precise number.

Without loss of generality, we consider for simplicity only processes with $Q_0 = \{q_0\}$, as multiple initial states can be encoded through local transitions from q_0. Given a process $\mathcal{P} = \langle Q, \Sigma, R, \{q_0\}\rangle$ and a cardinality constraint φ over $\mathcal{P}$ we denote by $\mathrm{val}(\varphi) \in \mathbb{N}$ the largest natural constant that appears in φ. We then denote by $\mathrm{psize}(\varphi)$ the natural $|Q| * \mathrm{val}(\varphi)$. Intuitively $\mathrm{psize}(\varphi)$ is the number of witness nodes we keep track of: we reserve $\mathrm{val}(\varphi)$ processes to each control state that may appear in φ.

A symbolic configuration for $\mathcal{P}$ and φ is then a pair $\theta = \langle v, S\rangle$ where $v \in Q^{\mathrm{psize}(\varphi)}$ is a vector of $\mathrm{psize}(\varphi)$ elements of Q and $S \subseteq Q$. For $q \in Q$, we then write $\#v(q)$ to indicate the number of occurrences of q in the vector v. Note that by definition $0 \leq \#v(q) \leq \mathrm{psize}(\varphi)$ for every $q \in Q$ and that $\sum_{q \in Q} \#v(q) = \mathrm{psize}(\varphi)$. This allows us to describe the set of configurations $[\![\theta]\!] \subseteq \Gamma$ characterized by a symbolic configuration $\theta = \langle v, S\rangle$ as follows: we have $\gamma \in [\![\theta]\!]$ if and only $\#\gamma(q) > \#v(q)$ for every $q \in S$ and $\#\gamma(q) = \#v(q)$ for every $q \in Q \setminus S$. Hence a symbolic configuration $\theta = \langle v, S\rangle$ represents all the configurations such that the number of occurrences of a control state q is greater than the number of occurrences of q in v if $q \in S$, or equal when $q \notin S$. We will say that a symbolic configuration θ satisfies the cardinality constraint φ, written $\theta \models \varphi$, iff $\gamma \models \varphi$ for all $\gamma \in [\![\theta]\!]$. We use Θ to represent the set of symbolic configurations.

We make the following non restrictive assumptions: there is no constraint on the unique initial state q_0 in the cardinality constraints, the only outgoing transitions from the state q_0

are local transitions (labelled with τ), in the symbolic configurations $\langle v, S\rangle$ we always have $q_0 \in S$, and the initial configuration θ_0 is $\langle(q_0, \ldots, q_0), \{q_0\}\rangle$. The most important assumption is the first one about the absence of constraints on q_0: it is needed to guarantee the correctness of our symbolic procedure. For instance, consider a process $\mathcal{P} = \langle\{q_0\}, \Sigma, R, \{q_0\}\rangle$ and a cardinality constraint φ of the form $1 \leq \#q_0 < 2$. We have then $\text{psize}(\varphi) = 2$ and the symbolic configurations are of the form $\langle(q_0, q_0), S\rangle$. It is then obvious that all the symbolic configurations do not satisfy φ while the initial concrete configuration with only one node does. The above assumptions are not restrictive because given a process $\mathcal{P} = \langle Q, \Sigma, R, \{q_0\}\rangle$ and a cardinality constraint φ, we can define a new process $\mathcal{P}' = \langle Q', \Sigma, R', \{q_{init}\}\rangle$ where $Q' = Q \cup \{q_{init}\}$ and $R' = R \cup \{\langle q_{init}, \tau, q_0\rangle\}$, i.e. q_{init} is a new initial state from which the process is enabled to go to q_0 thanks to a local transition. As there is no constraint in φ about q_{init} it is immediate to prove the following Lemma:

▶ **Lemma 5.2.** *$\mathcal{P} \models \Diamond\varphi$ if and only if $\mathcal{P}' \models \Diamond\varphi$.*

We now define a relation on the symbolic configurations to represent the effect that process rules have on symbolic configurations. Let $\mathcal{P} = \langle Q, \Sigma, R, \{q_0\}\rangle$ be a process, φ a cardinality constraint and θ the associated set of symbolic configurations. For each rule $r \in R$ of form $\langle q, !!a, q'\rangle$, we define the symbolic transition relation $\rightsquigarrow_r \subseteq \Theta \times \Theta$ as follows, we have $\langle v, S\rangle \rightsquigarrow_r \langle v', S'\rangle$ if and only if at least one of the two following conditions holds:

1. *(broadcast from a state in v)* there exists $i \in \{1, \ldots, \text{psize}(\varphi)\}$ such that $v[i] = q$ and $v'[i] = q'$ (i.e. the sending process switches state according to r) and:
   - for all $j \in \{1, \ldots, \text{psize}(\varphi)\}\backslash\{i\}$ we have either $v[j] = v'[j]$ or there exists $\langle q_r, ??a, q'_r\rangle \in R$ such that $v[j] = q_r$ and $v'[j] = q'_r$ (i.e. other processes in the pool may or may not react to the broadcast);
   - for each $q_s \in Q \setminus \{q_0\}$:
     - if $q_s \in S' \setminus S$ then there exists $q'_s \in S$ and $\langle q'_s, ??a, q_s\rangle \in R$,
     - if $q_s \in S \setminus S'$ then there exists $q'_s \in S'$ and $\langle q_s, ??a, q'_s\rangle \in R$.
2. *(broadcast from a state in S)* we have $q \in S$ and $q' \in S'$ (note that we could have that $q \in S'$ or $q \notin S'$), and the following conditions hold:
   - for all $j \in \{1, \ldots, \text{psize}(\varphi)\}$ we have either $v[j] = v'[j]$ or there exists $\langle q_r, ??a, q'_r\rangle \in R$ such that $v[j] = q_r \wedge v'[j] = q'_r$;
   - for each $q_s \in Q \setminus \{q, q'\}$, we have:
     - if $q_s \in S' \setminus S$ then there exists $\langle q'_s, ??a, q_s\rangle \in R$ with $q'_s \in S$,
     - if $q_s \in S \setminus S'$ then there exists $\langle q_s, ??a, q'_s\rangle \in R$ with $q'_s \in S'$.

We denote by $\rightsquigarrow \subseteq \Theta \times \Theta$ the relation such that $\theta \rightsquigarrow \theta'$ if and only if there exists a rule $r \in R$ such that $\theta \rightsquigarrow_r \theta'$, and $\rightsquigarrow^*$ represents its reflexive and transitive closure. The intuition behind this construction is that we do not perform any abstraction on the states present in the vector v but only on the states present in S, this because the states present in v are used as witnesses to satisfy the cardinality constraint φ.

As an example, for $\text{psize}(\varphi) = 5$, let $\langle(q_1, q_2, q_0, q_0, q_0), \{q_0, q_1, q_2\}\rangle$ be a symbolic configuration, and $\langle q_1, !!a, q'_1\rangle$ and $\langle q_2, ??a, q'_2\rangle$ be two transition rules. With a broadcast from a process in the vector we may reach, among others, $\langle(q'_1, q_2, q_0, q_0, q_0), \{q_0, q_1, q_2\}\rangle$, $\langle(q'_1, q'_2, q_0, q_0, q_0), \{q_0, q_1, q_2, q'_2\}\rangle$, or $\langle(q'_1, q'_2, q_0, q_0, q_0), \{q_0, q_1, q'_2\}\rangle$, whereas a broadcast from a process in the set may lead to $\langle(q_1, q_2, q_0, q_0, q_0), \{q_0, q_1, q_2, q'_1\}\rangle$, $\langle(q_1, q_2, q_0, q_0, q_0), \{q_0, q_1, q'_1, q'_2\}\rangle$, $\langle(q_1, q'_2, q_0, q_0, q_0), \{q_0, q_1, q_2, q'_1, q'_2\}\rangle$, or $\langle(q_1, q'_2, q_0, q_0, q_0), \{q_0, q'_1, q'_2\}\rangle$.

We will now prove that the symbolic configurations are well-suited to solve CRP. First, we show that if a symbolic configuration which satisfies φ is reachable from the initial symbolic

configuration, then there is a concrete configuration reachable from an initial configuration in γ_0 which also satisfies φ. This ensures a sound reasoning on symbolic configurations.

▶ **Lemma 5.3.** *If there exists $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$ and $\theta \models \varphi$, then $\mathcal{P} \models \Diamond\varphi$.*

Sketch of proof. For a symbolic $\theta = \langle v, S\rangle$ in Θ and $N \in \mathbb{N}$, we denote by $[\![\theta]\!]_N = \{\gamma \in [\![\theta]\!] \mid \forall q \in S.\#\gamma(q) > (N + \#v(q))\}$, i.e. the set of configurations which belong to $[\![\theta]\!]$ in which for each $q \in S$, there are at least N vertices (in addition to those already in the vector v). Note that with this definition $[\![\theta]\!]_0 = [\![\theta]\!]$. We then can prove the following property: given $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$, there exists $N \in \mathbb{N}$ such that for all $\gamma \in [\![\theta]\!]_N$, there exists an initial configuration $\gamma_0 \in \Gamma_0$ such that $\gamma_0 \rightarrow^* \gamma$. To show that this property is true, we reason by induction on the length of the execution choosing the N adequately at each step of the induction. Then if there exists $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$ and $\theta \models \varphi$, then there exists $\gamma_0 \in \theta_0$ and $\gamma \in [\![\theta]\!]$ such that $\gamma_0 \rightarrow^* \gamma$, and by the definition of $\models$ for symbolic configuration we deduce also that $\gamma \models \varphi$. Hence $\mathcal{P} \models \Diamond\varphi$. ◀

We will now show that a reasoning on symbolic configurations leads to completeness, in other words that if there is a reachable configuration that satisfies the cardinality constraint φ, then there is a reachable symbolic configuration that satisfies φ.

▶ **Lemma 5.4.** *If $\mathcal{P} \models \Diamond\varphi$, then there exists $\theta \in \Theta$ such that $\theta_0 \rightsquigarrow^* \theta$ and $\theta \models \varphi$.*

Sketch of proof. In order to prove this Lemma, we need to introduce some auxiliary notations. Given a configuration $\gamma \in \Gamma$, we define $\uparrow_{q_0} \gamma$ as the set $\{\gamma' \in \Gamma \mid \forall q \in Q \setminus \{q_0\}.\ \#\gamma'(q) = \#\gamma(q)\}$. The above definition is needed because we could reach a configuration γ which does not have enough processes to be represented by a symbolic configuration, but we can complete it by adding new vertices labelled by the initial state q_0 in order to solve the problem. We can then prove the following property by induction on the length of the concrete execution: for $\gamma_0 \in \Gamma_0$ and $\gamma \in \Gamma$ such that $\gamma_0 \rightarrow^* \gamma$, for all $\theta \in \Theta$ verifying $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$, we have $\theta_0 \rightsquigarrow^* \theta$. Basically, this property stipulates that given a reachable configuration γ, each symbolic configuration θ whose semantics $[\![\theta]\!]$ contains γ (modulo processes in state q_0) is also reachable.

The next step consists in proving that if $\gamma \in \Gamma$ is a configuration satisfying $\gamma \models \varphi$ then there exists $\theta \in \Theta$ such that $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$ and $\theta \models \varphi$. This can be proved providing an algorithm that builds $\theta = \langle v, S\rangle$ such that, for each $q \in Q$, either the processes in state q can be exactly represented within v only when $\#\gamma(q) \leq \mathrm{val}(\varphi)$, or $\#v(q) = \mathrm{val}(\varphi)$ and $q \in S$ when $\#\gamma(q) > \mathrm{val}(\varphi)$ (i.e. v is not large enough, recall that, apart for the states q_0 used to fill the "holes" in v, we reserve only up to $\mathrm{val}(\varphi)$ processes per state in v). Consider, e.g., a process with states $Q = \{q_0, q_1, q_2\}$, the formula $\varphi = 0 \leq \#q_1 < 3 \wedge 1 \leq \#q_2 < +\infty$ and the configuration with five processes $\gamma = \langle q_1, q_2, q_2, q_2, q_2\rangle$ such that $\gamma \models \varphi$. The symbolic configuration θ obtained is then $\langle (q_1, q_2, q_2, q_2, q_0, q_0, q_0, q_0, q_0), \{q_0, q_2\}\rangle$.

Since $\mathcal{P} \models \Diamond\varphi$, there exists an initial configuration $\gamma_0 \in \Gamma_0$ and a configuration $\gamma \in \Gamma$ such that $\gamma_0 \rightarrow^* \gamma$ and $\gamma \models \varphi$. By the second property we know there exists $\theta \in \Theta$ such that $\uparrow_{q_0} \gamma \cap [\![\theta]\!] \neq \emptyset$ and $\theta \models \varphi$, and the first property allows us to say that $\theta_0 \rightsquigarrow^* \theta$. ◀

We will now explain why CRP is in PSpace. The main idea is that we can reason on the graph of symbolic configurations. Note that by definition, since $\Theta = Q^{\mathrm{psize}(\varphi)} \times 2^Q$, the total number of symbolic configurations is $|\Theta| = |Q|^{\mathrm{psize}(\varphi)} * 2^{|Q|}$. Furthermore, checking whether a symbolic configuration satisfies a cardinality constraint can be done in PTime and checking whether two symbolic configurations belong to the symbolic transition relation $\rightsquigarrow$ can also be done in PTime. The PSpace algorithm (which is in reality an NPSpace algorithm) at each step guesses a new symbolic configuration, checks whether it is reachable from the previous guessed one and verifies whether it satisfies φ. When it encounters a symbolic

configuration that satisfies φ, it returns that $\mathcal{P} \models \Diamond\varphi$. Note that this algorithm needs only to store the number of configurations it has seen until now, and when this number reaches $|Q|^{\text{psize}(\varphi)} * 2^{|Q|}$, it means that the algorithm have seen all the symbolic configurations. Hence to store this number and the current and next symbolic configurations, the algorithm needs polynomial space (a number smaller than $|Q|^{\text{psize}(\varphi)} * 2^{|Q|}$ can be stored into a counter which requires at most $\text{psize}(\varphi) * \log(|Q|) + |Q| \log(2)$ space). Finally, lemmas 5.3 and 5.4 ensure us that such an algorithm is sound and complete and from Proposition 5.1 we have also a lower bound for CRP. Hence we deduce the main result of this paper.

▶ **Theorem 5.5.** *CRP is* PSPACE*-complete.*

References

1 Abdulla, P. A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. 127(2): 91–101 (1996)
2 Cheng, A., Esparza, J., Palsberg, J.: Complexity Results for 1-Safe Nets. Theor. Comput. Sci. 147(1&2): 117-136 (1995)
3 Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of Ad Hoc Networks. CONCUR'10: 313–327
4 Delzanno, G., Sangnier, A., Zavattaro, G.: On the Power of Cliques in the Parameterized verification of Ad Hoc Networks. FOSSACS'11: 441–455
5 Delzanno, G., Sangnier, A., Zavattaro, G.: Verification of Ad Hoc Networks with Node and Communication Failures. FORTE'12: 235-250
6 Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks. Research Report hal-00740518, HAL, CNRS, France, 2012
7 Ene, C., Muntean, T.: A broadcast based calculus for Communicating Systems. IPDPS'01: 149
8 Esparza, J., Finkel, A., Mayr, R.: On the verification of Broadcast Protocols. LICS'99: 352–359
9 Fehnker, A., van Hoesel, L., Mader, A.: Modelling and verification of the LMAC protocol for wireless sensor networks. IFM'07: 253–272
10 Godskesen, J.C.: A calculus for Mobile Ad Hoc Networks. COORDINATION'07: 132–150
11 S. Joshi, B. König: Applying the Graph Minor Theorem to the Verification of Graph Transformation Systems. CAV'08: 214-226
12 Ladner, R. E.: The circuit value problem is logspace complete for P. SIGACT News: 18–20 (1977)
13 Lipton R.J.: The Reachability Problem Requires Exponential Space. Department of Computer Science. Research Report. Yale University. (1976)
14 Merro, M.: An observational theory for Mobile Ad Hoc Networks. Inf. Comput. 207(2): 194–208 (2009)
15 Nanz, S., Hankin, C.: A Framework for security analysis of mobile wireless networks. TCS, 367(1–2):203-227 (2006)
16 Prasad, K.V.S.: A Calculus of Broadcasting Systems. SCP, 25(2–3): 285–327 (1995)
17 Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of Ad Hoc Routing Protocols. TACAS'08: 18–32
18 Singh, A., Ramakrishnan, C. R., Smolka, S. A.: A process calculus for Mobile Ad Hoc Networks. COORDINATION'08: 296–314
19 Singh, A., Ramakrishnan, C. R., Smolka, S. A.: Query-Based model checking of Ad Hoc Network Protocols. CONCUR '09: 603–619
20 Rackoff C.: The Covering and Boundedness Problems for Vector Addition Systems. TCS, 6:223-231 (1978)

Extending the Rackoff technique to Affine nets

Rémi Bonnet*, **Alain Finkel**†, **and M. Praveen**‡

LSV, INRIA, CNRS, ENS Cachan

Abstract

We study the possibility of extending the Rackoff technique to Affine nets, which are Petri nets extended with affine functions. The Rackoff technique has been used for establishing EXPSPACE upper bounds for the coverability and boundedness problems for Petri nets. We show that this technique can be extended to strongly increasing Affine nets, obtaining better upper bounds compared to known results. The possible copies between places of a strongly increasing Affine net make this extension non-trivial. One cannot expect similar results for the entire class of Affine nets since coverability is Ackermann-hard and boundedness is undecidable. Moreover, it can be proved that model checking a logic expressing generalized coverability properties is undecidable for strongly increasing Affine nets, while it is known to be EXPSPACE-complete for Petri nets.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, D.2.4 Software/Program Verification, F.4.1 Mathematical Logic

Keywords and phrases Complexity of VASS, Affine nets, Rackoff technique, model checking, coverability, boundedness.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.301

1 Introduction

Context Petri nets are infinite state models and have been used for modelling and verifying properties of concurrent systems. Various extensions of Petri nets that increase the power of transitions have been studied, for example Reset/Transfer (Petri) nets [8], Self-Modifying nets [21] and Petri nets with inhibitory arcs [17]. In [9], Well Structured nets are defined as another extension where transitions can be any non-decreasing function. The same paper also defines Affine Well Structured nets (shortly: Affine nets) that can be seen as the affine restriction of Well Structured nets, or as a restriction of the Self-Modifying nets of [21] to matrices with only non-negative integers.

While reachability is decidable for Petri Nets [11, 15, 13], it is undecidable for extensions with at least two extended transitions like Double/Reset/Transfer/Zero-test arcs [8]. However, it remains decidable for Petri Nets with one such extended arc [2] or even with hierarchical zero-tests [17]. The framework of Well-Structured Transition Systems [10] provides the decidability of coverability, termination and boundedness for Petri nets and some of its monotonic extensions [8, 9]. However, boundedness is undecidable for Reset nets (and hence for Affine nets) [8]. Complexity results on Petri nets extensions are scarce, two notable results being that coverability is Ackermann-complete for Reset nets [20, 19] (while reachability

* This author is supported by the french Agence Nationale de la Recherche, REACHARD (grant ANR-11-BS02-001)

† This author is supported by the french Agence Nationale de la Recherche, REACHARD (grant ANR-11-BS02-001) and by the "Chaire DIGITEO"

‡ This author is supported by an ERCIM "Alain Bensoussan" Fellowship Programme. This programme is supported by the Marie Curie Co-funding of Regional, National and International Programmes (COFUND) of the European Commission.

and boundedness are undecidable) and boundedness is EXPSPACE-complete for a subclass of (strongly increasing) Affine nets [7].

Other extensions of Petri nets increase the state space of the transition system. These are for example branching Vector Addition Systems [6], ν-Petri nets [18], or Data nets [12] (equivalent to Timed Petri nets [3]). As ν-Petri nets and Data nets subsume Reset nets, boundedness and reachability are undecidable, and coverability is Ackermann-hard. On the other hand, while reachability is an (important) open problem for branching Vector Addition Systems, coverability and boundedness are known to be ALTEXPSPACE-complete by a proof that uses the Rackoff technique [6].

Finally, we note that some recent papers [5, 1] have extended the Rackoff technique to show EXPSPACE upper bounds for the model-checking of some logics (that generalizes the notion of coverability and boundedness) for Petri nets.

Our contribution The goal is to exhibit a class of extensions of Petri nets for which the Rackoff technique can be extended in order to give an EXPSPACE upper bound for coverability and boundedness. We do not look at extensions that change state spaces, as the complexity of coverability and boundedness for those is known to be either ALTEXPSPACE-complete (branching Vector Addition Systems) or Ackermann-hard (ν-Petri nets, Data nets). Moreover, as this technique relies heavily on the monotonicity of Petri Nets, it is natural to consider only monotonic extensions. The largest classes of such extensions are Affine nets and Well Structured nets, that both include most of the usually studied Petri net extensions. As we know that coverability and boundedness are respectively Ackermann-hard and undecidable for Reset nets, we must forbid resets in order for our generalization to work. This is done by disallowing any 0 in the diagonal of the matrices associated with the functions of Affine nets, yielding again the class of *strongly increasing Affine nets*, as defined in [9], that are equivalent to the Post-Self-Modifying nets (PSM) defined by Valk in [21]. This class is interesting because it strictly subsumes Petri nets. For example, PSM can recognize the language of palindromes, which Petri nets can not. More generally, all recursively enumerable languages are recognized by PSM [21], while boundedness (and other properties) is still decidable [21].

While the complexity of the reachability problem for Petri nets is unknown, the complexity of coverability and boundedness has been shown to be EXPSPACE-complete (lower bound of $\text{SPACE}(\mathcal{O}(2^{c\sqrt{n}}))$ by Lipton [14] and $\text{SPACE}(\mathcal{O}(2^{cn \log n}))$ upper bound by Rackoff [16], where n is the size of the net). In [7], the boundedness problem is shown to be in $\text{SPACE}(\mathcal{O}(2^{cn^2 \log n}))$ for Post-Self-Modifying nets: the proof associates a standard Petri net that weakly simulates the original Post-Self-Modifying net and then applies the Rackoff theorem [16] as a black box (EXPSPACE upper bound for coverability could also be shown by the same construction).

We give two results: (1) We extend the Rackoff technique to work directly on strongly increasing Affine nets, improving the upper bounds for coverability and boundedness (from $\text{SPACE}(\mathcal{O}(2^{cn^2 \log n}))$ to $\text{SPACE}(\mathcal{O}(2^{cn \log n}))$). (2) We state the limit of strongly increasing Affine nets by proving that model checking a fragment of CTL (which can express generalizations of boundedness and various other problems) is undecidable for strongly increasing Affine nets, while it is EXPSPACE-complete for Petri nets [1].

Following are the three main difficulties in extending the Rackoff technique to strongly increasing Affine nets.

1. Showing upper bounds for the lengths of sequences certifying coverability or unboundedness is not enough — short sequences can give rise to large numbers.
2. We can no longer rely on ignoring places that go above some value. The effect of a

transition on a place will depend on the exact value at other places.

3. The effect of firing a sequence of transitions can not be determined by its Parikh image.

To overcome the first difficulty, we define transition systems that abstract the real ones, where markings from short sequences of transitions will have either small numbers or ω. This also overcomes the second difficulty, since ignored places will have the value ω in the abstract transition systems. If such a place affects another place, the affected place will also get the value ω. To overcome the third difficulty, we classify the set of places according to the way they affect each other. Places that have high values and interfere among themselves will always be unbounded so that they can be "ignored" (implemented by introducing another abstraction), leaving behind places that are amenable to analysis by Petri net techniques. Since this technique depends on the observation that places interfering with one another are unbounded, it can not be used for problems that require more precise answers than unboundedness. Model checking the fragment of CTL mentioned above does require such precise answers and turns out to be undecidable for strongly increasing Affine nets.

The following table summarises the complexity of various problems on Petri nets and strongly increasing Affine nets, with the contributions of this paper in bold. Abbreviations used in the table: SIAN for Strongly increasing Affine nets, MC(eiPrCTL) for model checking eventually increasing Presburger CTL, $\textsc{Sp}(2^{c\sqrt{n}} : 2^{cn \log n})$ for $\textsc{Space}(2^{c\sqrt{n}})$ lower bound and $\textsc{Space}(2^{cn \log n})$ upper bound.

	Petri nets	SIAN	Affine nets
Reachability	Decidable [15, 11]	Undecidable [8]	Undecidable [8]
Coverability	$\textsc{Sp}(2^{c\sqrt{n}} : 2^{cn \log n})$ [14, 16]	$\textsc{Sp}(2^{c\sqrt{n}} : \mathbf{2^{cn \log n}})$	Ackermann-hard [20]
Boundedness	$\textsc{Sp}(2^{c\sqrt{n}} : 2^{cn \log n})$ [14, 16]	$\textsc{Sp}(2^{c\sqrt{n}} : \mathbf{2^{cn \log n}})$	Undecidable [8]
MC(eiPrCTL)	Expspace-complete [1]	**Undecidable**	Undecidable [8]

Due to space constraints, proofs have been skipped in the following, as are the details of model checking eventually increasing Presburger CTL. All missing details and proofs can be found in the full version of this paper [4].

2 Preliminaries

Let $\mathbb{Z}$ be the set of integers, $\mathbb{N}$ be the set of non-negative integers and $\mathbb{N}^+$ be the set of positive integers. For any set P, $card(P)$ is the cardinality of P.

A *transition system* $\mathcal{S} = (S, \to)$ is a set S endowed with a transition relation "$\to$", i.e., with a binary relation on the set S. We write $s \xrightarrow{+} t$ to mean that there exist $r \in \mathbb{N}^+$ and a sequence of states $s_0 = s, s_1, \ldots, s_r = t$ such that $s_0 \to s_1 \to \cdots \to s_r$. We write $s \xrightarrow{*} t$ to mean that $s \xrightarrow{+} t$ or $s = t$. A state $t \in S$ of a transition system $\mathcal{S} = (S, \to)$ is *reachable* from a state s if $s \xrightarrow{*} t$. The *reachability set* of $\mathcal{S}$ from the state s_0 is denoted by $RS(\mathcal{S}, s_0)$ and is defined to be the set of states reachable from s_0.

Let P be a finite non-empty set of *places* with $card(P) = m \in \mathbb{N}^+$ and let $\langle p_1, \ldots, p_m \rangle$ be an arbitrary but fixed order on the set of places. A function $M : P \to \mathbb{N}$ is called a *marking*. We denote by $\mathbf{0}$ the marking such that $\mathbf{0}(p) = 0$ for all $p \in P$. Given a subset $Q \subseteq P$ and markings M_1, M_2, we write $M_1 =_Q M_2$ (resp. $M_1 \geq_Q M_2$) if $M_1(p) = M_2(p)$ (resp. $M_1(p) \geq M_2(p)$) for all $p \in Q$. We write $M_1 > M_2$ if $M_1 \geq M_2$ and $M_1 \neq M_2$. We denote by Id the identity matrix, whose dimension will be clear from context. We denote by $\mathbf{A}_1 \geq \mathbf{A}_2$, where $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{N}^{m \times m}$, the condition that $\mathbf{A}_1(p_1, p_2) \geq \mathbf{A}_2(p_1, p_2)$ for all $p_1, p_2 \in P$. We consider *(positive) affine functions* from $\mathbb{N}^m$ into $\mathbb{N}^m$ defined by $f(M) = \mathbf{A}M + \mathbf{B}$, where $\mathbf{A}$ is a (positive) matrix in $\mathbb{N}^{m \times m}$ and $\mathbf{B}$ is a vector in $\mathbb{Z}^m$. It can be verified that for every

affine function $f(M) = \mathbf{A}M + \mathbf{B}$ with an upward closed domain (i.e., $domf \subseteq \mathbb{N}^P$ such that $M_1 \in domf$ and $M_1 \leq M_2$ imply $M_2 \in domf$), there exists a *finite* set of vectors $\{\mathbf{C}_1, \ldots, \mathbf{C}_k\} \subseteq \mathbb{Z}^m$ such that $domf = \cup_{1 \leq i \leq k}\{M \in \mathbb{N}^m \mid \mathbf{A}M + \mathbf{B} \geq \mathbf{0} \text{ and } M + \mathbf{C}_i \geq \mathbf{0}\}$. With $\mathbf{A} \in \mathbb{N}^{m \times m}$ and $\mathbf{B}, \mathbf{C} \in \mathbb{Z}^m$, we denote by $f \triangleq (\mathbf{A}, \mathbf{B}, \mathbf{C})$ the affine function such that $f(M) = \mathbf{A}M + \mathbf{B}$ and $domf = \{M \in \mathbb{N}^m \mid \mathbf{A}M + \mathbf{B} \geq \mathbf{0} \text{ and } M + \mathbf{C} \geq \mathbf{0}\}$. In terms of Petri nets, the vector $\mathbf{C}$ restricts the markings to which the transition can be applied. For example, if the transition should not subtract anything from a place p but should only be applicable to markings M with $M(p) \geq 1$, we can set $\mathbf{A}(p, p) = 1, \mathbf{B}(p) = 0$ and $\mathbf{C}(p) = -1$. In the following, we just write $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ if the name f is not important.

▶ **Definition 1.** An *Affine net* $\mathcal{N}$ (of dimension m) is a tuple $\mathcal{N} = (m, F)$ where $m \in \mathbb{N}^+$ and F is a finite set of affine functions with upward closed domains in $\mathbb{N}^m$.

The application of the transition function f to M_1 resulting in M_2 is denoted by $M_1 \xrightarrow{f} M_2$. The associated *Affine transition system* $\mathcal{S}_\mathcal{N} = (S, \xrightarrow{F})$ is naturally defined by $S = \mathbb{N}^P$ and $M_1 \xrightarrow{f} M_2$ iff $M_1 \in domf$ and $f(M_1) = M_2$. If there is a sequence $\sigma = f_1 f_2 \cdots f_r$ of transition functions such that $M \xrightarrow{f_1} M_1 \xrightarrow{f_2} \cdots\cdots \xrightarrow{f_r} M_r$, we denote it by $M \xrightarrow{\sigma} M_r$. The markings $M, M_1, \ldots, M_r$ are called *intermediate markings arising while firing* σ *from* M. We say a sequence σ of transition functions is *enabled at a marking* M if $M \xrightarrow{\sigma} M'$ for some marking M'. We denote the length of σ by $|\sigma|$. We denote the set of transition functions of $\mathcal{N}$ occurring in σ by $alph(\sigma)$. A sequence σ' is called a *sub-sequence* of σ if σ' can be obtained from σ by removing some transition functions.

▶ **Definition 2.** An affine function $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ is *strongly increasing* [9, Section 2.2] if $\mathbf{A} \geq Id$. An Affine net $\mathcal{N} = (m, F)$ is strongly increasing if each of its functions is strongly increasing.

Note that if $M_1 \xrightarrow{f} M_2$ and $M_1' > M_1$, then the fact that f is strongly increasing implies that $M_1' \xrightarrow{f} M_2'$ for some $M_2' > M_2$ and for every $p \in P$, $M_1'(p) > M_1(p)$ implies $M_2'(p) > M_2(p)$.

▶ **Definition 3.** Given an Affine net $\mathcal{N}$ with an initial marking M_{init} and a target marking M_{cov}, the *coverability problem* is to determine if there exists a marking $M \in RS(\mathcal{S}_\mathcal{N}, M_{init})$ such that $M \geq M_{cov}$. The *boundedness problem* is to determine if there exists a number $B \in \mathbb{N}$ such that for all markings $M \in RS(\mathcal{S}_\mathcal{N}, M_{init})$, $M(p) \leq B$ for all $p \in P$.

For an Affine net $\mathcal{N}$, $R_\mathcal{N}$ will denote the maximum absolute value of any entry in $\mathbf{A}$, $\mathbf{B}$ or $\mathbf{C}$ for any transition function $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ of $\mathcal{N}$. When $\mathcal{N}$ is clear from context, we skip the subscript $\mathcal{N}$ and write R. We also write *function* instead of *transition function* when it is clear from context that it is a transition function in an Affine net. The *size* of $\mathcal{N}$ with initial marking M_{init} is defined to be $(card(F)(m^2 + m) \log R + m \log \|M_{init}\|_\infty)$, where $\|M_{init}\|_\infty$ is the maximum entry in M_{init}. If $\mathbf{A} = Id$ for each function $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ of $\mathcal{N}$, then $\mathcal{N}$ is a Petri net.

In Affine nets, markings cannot decrease too much.

▶ Proposition 4. If $M_1 \xrightarrow{\sigma} M_2$, then $M_2(p) \geq M_1(p) - R|\sigma|$ for all $p \in P$.

3 Value Abstracted Semantics

In Affine nets, a short sequence of functions can generate markings with large values. Beyond some value, it is not necessary to store the exact values of a marking to decide coverability and boundedness. Let $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$ and $\mathbb{Z}_\omega = \mathbb{Z} \cup \{\omega\}$ where addition, multiplication

and order are as usual with the extra definition of $\omega \times 0 = 0 \times \omega = 0$. To avoid using excessive memory space to store large values of markings, we introduce extended markings $W : P \to \mathbb{N}_\omega$. The domains of transitions functions are extended to include extended markings: for a function $f = (\mathbf{A}, \mathbf{B}, \mathbf{C})$ and an extended marking W, we have $W \in dom f$ iff $W + \mathbf{C} \geq \mathbf{0}$ and $\mathbf{A}W + \mathbf{B} \geq \mathbf{0}$. The result W' of applying f to $W \in dom f$ is given by $W' = \mathbf{A}W + \mathbf{B}$, denoted by $W \xrightarrow{f} W'$.

For an extended marking $W : P \to \mathbb{N}_\omega$, let $\omega(W) = \{p \in P \mid W(p) = \omega\}$ and $\overline{\omega(W)} = P \setminus \omega(W)$. For a function $t : \{0, \ldots, m\} \to \mathbb{N}$ (which will be used to denote thresholds) and an extended marking W, we define $[W]_{t \to \omega}$ and $[W]_{\omega \to t}$ by:

$$([W]_{t\to\omega})(p) = \begin{cases} W(p) & \text{if } W(p) < t(card(\overline{\omega(W)})), \\ \omega & \text{otherwise.} \end{cases}$$

$$([W]_{\omega\to t})(p) = \begin{cases} W(p) & \text{if } W(p) \in \mathbb{N}, \\ t(card(\overline{\omega(W)}) + 1) & \text{otherwise.} \end{cases}$$

The threshold function t gives the threshold beyond which numbers can be abstracted. In the extended marking $[W]_{t\to\omega}$, values beyond the threshold given by t are abstracted by ω. In the marking $[W]_{\omega\to t}$, abstraction is reversed by replacing ω with the corresponding threshold value.

▶ **Definition 5.** Let $t : \{0, \ldots, m\} \to \mathbb{N}$ be a threshold function and $\mathcal{N}$ be a strongly increasing Affine net. The associated *t-transition system* $\mathcal{S}_{\mathcal{N},t} = (S_t, \xrightarrow{F}_t)$ is defined by $S_t = \mathbb{N}_\omega^P$ and $W_1 \xrightarrow{(\mathbf{A},\mathbf{B},\mathbf{C})}_t W_2$ iff $W_1 \geq \mathbf{C}$ and $W_2 = [(\mathbf{A}W_1 + \mathbf{B})]_{t\to\omega} \in \mathbb{N}_\omega^P$. We write $W_0 \xrightarrow{\sigma}_t W_r$ if $\sigma = f_1 \cdots f_r$ and $W_{i-1} \xrightarrow{f_i}_t W_i$ for each i between 1 and r. The extended markings $W_0, \ldots, W_r$ are called *intermediate extended markings in the run* $W_0 \xrightarrow{\sigma}_t W_r$.

Note that for any $W_1 \xrightarrow{f}_t W_2$, $\omega(W_2) \supseteq \omega(W_1)$. In the t-transition system, a place having the value ω will retain it after the application of any function. The following propositions establish some relationships between t-transition systems and natural transition systems.

▶ Proposition 6. Let $W_1 \xrightarrow{\sigma}_t W_2$, $card(\overline{\omega(W_2)}) = card(\overline{\omega(W_1)}) < m$ and $t(card(\overline{\omega(W_1)}) + 1) \geq R|\sigma| + x$ for some $x \in \mathbb{N}$. Then $[W_1]_{\omega\to t} \xrightarrow{\sigma} M_2$ such that $M_2 =_{\overline{\omega(W_2)}} W_2$ and $M_2(p) \geq x$ for all $p \in \omega(W_2)$.

A routine induction on $|\sigma|$ allows to prove the following:

▶ Proposition 7. If $M_1 \xrightarrow{\sigma} M_2$, then $M_1' \xrightarrow{\sigma}_t W_2 \geq M_2$ for any $M_1' \geq M_1$.

4 Coverability

In this section, we give a SPACE($\mathcal{O}(2^{c_2 n \log n})$) upper bound for the coverability problem in strongly increasing Affine nets, for some constant c_2. Let $R' = \max(\{M_{cov}(p) \mid p \in P\} \cup \{R\})$, where M_{cov} is the marking to be covered. In the rest of this section, we fix a strongly increasing Affine net $\mathcal{N} = (m, F)$ with an initial marking M_{init} and the marking to be covered M_{cov}. The set of places is $P = \{p_1, \ldots, p_m\}$.

We briefly recall the Rackoff technique for the EXPSPACE upper bound for the coverability problem in Petri nets. The idea is to define a function $\ell : \mathbb{N} \to \mathbb{N}$ and prove that for a Petri net with m places, coverable markings can be covered with sequences of transitions of length at most $\ell(m)$. This is done by induction on the number of places. In a Petri net with $i + 1$

places, suppose $M_{init} \xrightarrow{\sigma} M' \geq M_{cov}$ and M is the first intermediate marking where one of the values is more than $R\ell(i) + R' - 1$ (this is the intuition behind the definition of the threshold function t_1 below). If there is no such marking, all intermediate markings have small values and it is easy to bound the length of σ by $(R\ell(i) + R')^{i+1}$. Otherwise, let $\sigma = \sigma_1\sigma_2$ such that $M_{init} \xrightarrow{\sigma_1} M \xrightarrow{\sigma_2} M' \geq M_{cov}$. The length of σ_1 is bounded by $(R\ell(i)+R')^{i+1}$. Temporarily forgetting the existence of place p (where $M(p) \geq R\ell(i)+R'$), we conclude by induction hypothesis that starting from M, M_{cov} can be covered (in all places except p) with a sequence σ_2' of length at most $\ell(i)$. Since $M(p) \geq R\ell(i) + R'$ and σ_2' reduces the value in p by at most $R\ell(i)$, σ_2' in fact covers all places, including p. Hence, $\sigma_1\sigma_2'$ covers M_{cov} from M_{init} and its length is at most $(R\ell(i) + R')^{i+1} + \ell(i) + 1$. This is the intuition behind the definition of the length function ℓ_1 below. The counterpart of "temporarily forgetting p" is assigning it the value ω.

▶ **Definition 8.** The functions $\ell_1, t_1 : \mathbb{N} \to \mathbb{N}$ are as follows.

$$t_1(0) = 0 \qquad \ell_1(0) = 0$$
$$t_1(i+1) = R\ell_1(i) + R' \qquad \ell_1(i+1) = (t_1(i+1))^{i+1} + \ell_1(i) + 1$$

▶ **Definition 9.** A *covering sequence enabled at* M is a sequence σ of functions such that $M \xrightarrow{\sigma} M'$ and $M' \geq M_{cov}$. A t_1*-covering sequence enabled at* W is a sequence σ of functions such that $W \xrightarrow{\sigma}_{t_1} W'$ and $W' \geq M_{cov}$.

The following lemma shows that even after abstracting values that are above the ones given by the threshold function t_1, there is still enough information to check coverability.

▶ **Lemma 10.** *If a* t_1*-covering sequence* σ *is enabled at* W*, then* M_{cov} *is coverable from* $[W]_{\omega \to t_1}$.

▶ **Lemma 11.** *If there is a covering sequence* σ *enabled at* M_{init}*, there is a* t_1*-covering sequence* σ' *enabled at* M_{init} *such that* $|\sigma'| \leq \ell_1(m)$ *(recall that* $m = card(P)$*).*

▶ **Lemma 12.** *For all* $i \in \mathbb{N}$, $\ell_1(i) \leq (6RR')^{(i+1)!}$.

▶ **Theorem 13.** *For some constant* c_1*, the coverability problem for strongly increasing Affine nets is in* NSPACE$(\mathcal{O}(2^{c_1 m \log m}(\log R' + \log\|M_{init}\|_\infty)))$.

Taking $n = (card(F)(m^2 + m) \log R + m \log\|M_{init}\|_\infty) + m \log\|M_{cov}\|_\infty$ as the size of the input to the coverability problem, we can infer from the above theorem an upper bound of SPACE$(\mathcal{O}(2^{c_2 n \log n}))$.

5 Boundedness

In this section, we give a SPACE$(\mathcal{O}(2^{c_4 n \log n}))$ upper bound for the boundedness problem in strongly increasing Affine nets, for some constant c_4. In the rest of this section, we fix a strongly increasing Affine net $\mathcal{N} = (m, F)$ with an initial marking M_{init}. The set of places is $P = \{p_1, \ldots, p_m\}$.

▶ **Definition 14.** A *self-covering pair enabled at* M is a pair (σ_1, σ_2) of sequences of functions such that $M \xrightarrow{\sigma_1} M_1 \xrightarrow{\sigma_2} M_2$ and $M_2 > M_1$.

Since all transition functions are strongly increasing and their domains are upward closed, we can infer from the above definition that $M_i \xrightarrow{\sigma_2} M_{i+1}$ and $M_{i+1} > M_i$ for all $i \in \mathbb{N}^+$. Hence, if a self covering pair is enabled at M_{init}, then $\mathcal{N}$ is unbounded. Conversely, if $\mathcal{N}$ is unbounded, infinitely many distinct markings can be reached from M_{init}. Since there

are only finitely many transition functions, König's lemma implies that there is an infinite sequence of functions enabled at M_{init} such that all intermediate markings are distinct. We infer from Dickson's lemma that there are two markings M_1, M_2 along this sequence such that M_2 is after M_1 and $M_2 > M_1$. Let $M_{init} \xrightarrow{\sigma_1} M_1 \xrightarrow{\sigma_2} M_2$. By Definition 14, (σ_1, σ_2) is a self-covering pair enabled at M_{init}.

The Rackoff technique again defines a length function $\ell' : \mathbb{N} \to \mathbb{N}$ and shows that if a Petri net with m places is unbounded, there is a self-covering pair of total length at most $\ell'(m)$. As an example, let us consider giving an upper bound for $\ell'(2)$. Consider the sequence of markings shown below, produced by a self-covering pair. Let σ_2 be the portion occurring

$$\begin{array}{lccccccccc}
p_1: & 1 & & 100 & & \left[\begin{array}{c}90\\140\\100\\200\\1\\3\end{array}\right. & & \begin{array}{c}190\\280\\90\\190\\1\\3\end{array}\left.\right] & & 180\\
p_2: & 2 & & 150 & & & & & & 270\\
p_3: & 1 & & 100 & & & & & & 100\\
p_4: & 3 & \cdots & 200 & \cdots & & \cdots & & \cdots & 200\\
p_5: & 0 & & 2 & & & & & & 3\\
p_6: & 1 & & 4 & & & & & & 4
\end{array}$$

after the first marking where p_1 has the value 100. Since p_5, p_6 have low values throught, we would like to abstract the remaining places and reduce the length of σ_2 to get an upper bound on $\ell'(2)$. We denote $\{p_5, p_6\}$ by $P^{\sigma_2}_{<\omega}$. In the block of intermediate markings shown above enclosed in [ ], the first and last markings are identical when projected to p_5 and p_6. Since this block does not change p_5 and p_6, we can remove this block, provided that after removal, the abstracted places p_1, p_2, p_3, p_4 will still have values at least $100, 150, 100, 200$ respectively. To decide whether this is the case, the effect of the block on p_1, p_2, p_3, p_4 is calculated in a Petri net by simply summing up the effect of each transition in the block. In a strongly increasing Affine net, this is however not possible since the effect of the block depends not only on the transitions in it, but also on the values in the marking at the beginning of the block. In addition, affine functions can copy the value of one place to another one.

If some transition copies the value of some place among p_1, p_2, p_3, p_4 into p_5 or p_6, a large value will result in p_5 or p_6, so that they too can be abstracted, letting us use induction hypothesis to deal with the remaining fewer number of non-abstracted places. To deal with the other case, we assume that no transition in σ_2 does this kind of copying (σ_2 isolates $\{p_5, p_6\}$ from $\{p_1, p_2, p_3, p_4\}$). Next, suppose a function $f = (\mathbf{A}, \mathbf{B}, \mathbf{C})$ occurs inside the block, where $\mathbf{A}$ and $\mathbf{B}$ are as shown in Fig. 1 in the next page. Let the rows and columns of $\mathbf{A}$ correspond to $p_1, p_2, \ldots, p_6$ in that order. The function f doubles the value in p_2 and copies the value of p_3 to p_1, but isolates $\{p_3, p_4\}$ from $\{p_1, p_2, p_3, p_4\}$. In the following, we will say f "crosses" $P^{\sigma_2}_{\times} = \{p_1, p_2\}$ and isolates $P^{\sigma_2}_{is} = \{p_3, p_4\}$ from $P \setminus P^{\sigma_2}_{<\omega} = \{p_1, p_2, p_3, p_4\}$. Since p_1, p_2 had large values to begin with, they will have even larger values after crossed by f. We will see that this will result in crossed places becoming unbounded, and so we can forget the exact effect of a block on such places, and just remember that they are crossed by some function. The forgetting part is done in Definition 16 by simplifying the matrix $\mathbf{A}$, and the remembering part is done by setting the corresponding entry in $\mathbf{B}$ to 1. To decide whether a block can be removed or not, it remains to compute the effect of $P^{\sigma_2}_{<\omega} = \{p_5, p_6\}$ on $P^{\sigma_2}_{is} = \{p_3, p_4\}$. This can be achieved since we know the exact values in p_5, p_6. This is formalised in the proof of Lemma 20.

▶ **Definition 15.** Let $f = (\mathbf{A}, \mathbf{B}, \mathbf{C})$ be a function.

- f *multiplies* a place p if $\mathbf{A}(p, p) \geq 2$.
- f *copies* p' *to* p if $\mathbf{A}(p, p') \geq 1$ for any two places $p \neq p'$.
- f *isolates* Q_1 *from* Q_2, where $Q_1, Q_2 \subseteq P$, if $\mathbf{A}(p, p') = 0$ for all $p \in Q_1$ and $p' \in Q_2 \setminus \{p\}$.

Although the sets $P^{\sigma_2}_{<\omega}$, $P^{\sigma_2}_{\times}$ and $P^{\sigma_2}_{is}$ are determined by σ_2, we avoid heavy notation in the following definition and instead use an arbitrary partition of P into $P_{<\omega}$, $P_{\times}$ and P_{is}.

▶ **Definition 16.** Let $\rho = (P_{<\omega}, P_{\times}, P_{is})$ be a triple such that the sets $P_{<\omega}$, $P_{\times}$ and P_{is} partition the set of places P. To each function $f = (\mathbf{A}, \mathbf{B}, \mathbf{C})$, we associate another function $f[\rho] = (\mathbf{A}[\rho], \mathbf{B}[\rho], \mathbf{C})$, where $\mathbf{A}[\rho]$ and $\mathbf{B}[\rho]$ are as follows.

$$(\mathbf{B}[\rho])(p) = \begin{cases} \mathbf{B}(p) & \text{if } p \notin P_{\times} \\ 1 & \text{if } f \text{ multiplies } p \in P_{\times} \\ 1 & \text{if } f \text{ copies } p' \text{ to } p \in P_{\times}, \\ & p' \in (P_{\times} \cup P_{is} \setminus \{p\}) \\ 0 & \text{otherwise} \end{cases} \quad (\mathbf{A}[\rho])(p, p') = \begin{cases} \mathbf{A}(p, p') & \text{if } p \notin P_{\times} \\ 1 & \text{if } p \in P_{\times} \\ & \text{and } p = p' \\ 0 & \text{otherwise} \end{cases}$$

Associated with the triple ρ is the *ρ-transition system* $\mathcal{S}_{\mathcal{N},\rho} = (S_\rho, \longrightarrow_\rho)$, defined by $S_\rho = \mathbb{N}^P_\omega$ and $W_1 \xrightarrow{(\mathbf{A},\mathbf{B},\mathbf{C})}_\rho W_2$ iff $W_1 + \mathbf{C} \geq \mathbf{0}$ and $W_2 = \mathbf{A}[\rho]W_1 + \mathbf{B}[\rho] \in \mathbb{N}^P_\omega$. We write $W_0 \xrightarrow{\sigma}_\rho W_r$ if $\sigma = f_1 \cdots f_r$ and $W_{i-1} \xrightarrow{f_i}_\rho W_i$ for each i between 1 and r. We write $\sigma[\rho]$ to denote the sequence of functions obtained from σ by replacing each function f of σ by $f[\rho]$.

For any extended marking W, if $W(p) \in \mathbb{N}$ for all $p \in P$ (i.e., if W is a marking), then applying any function to W in the ρ-transition system will result in another marking (new ω values are not introduced). In the example given before Definition 15, partition the set of places into $P_{\times} = \{p_1, p_2\}$, $P_{is} = \{p_3, p_4\}$ and $P_{<\omega} = \{p_5, p_6\}$. The corresponding matrices $\mathbf{A}[\rho]$ and $\mathbf{B}[\rho]$ defining $f[\rho]$ are as shown below.

Figure 1 Examples of transition functions

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 2 \\ 0 \\ -1 \end{bmatrix} \quad \mathbf{A}[\rho] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \mathbf{B}[\rho] = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 2 \\ 0 \\ -1 \end{bmatrix}$$

We obtain $\mathbf{A}[\rho]$ from $\mathbf{A}$ by replacing the first two rows by the corresponding rows of the identity matrix. The fact that $\{p_1, p_2\}$ is the set of places "crossed" by f is instead indicated by setting $\mathbf{B}[\rho](p_1) = \mathbf{B}[\rho](p_2) = 1$. The other entries of $\mathbf{B}[\rho]$ are not changed. Places with high values and crossed by some function will always be unbounded and this technique can not be applied for problems that need more precise answers. For example, eventually increasing existential Presburger CTL (eiPrECTL$_{\geq}$(U)), introduced in [1], can express the presence of sequences along which the value of one place grows unboundedly while another place remains bounded. Model checking eiPrECTL$_{\geq}$(U) is shown to be EXPSPACE-complete [1] for Petri nets by extending the Rackoff technique. We show in the full version that model checking eiPrECTL$_{\geq}$(U) is undecidable for strongly increasing Affine nets.

▶ **Definition 17.** Let $\rho = (P_{<\omega}, P_{\times}, P_{is})$ be a triple such that the sets $P_{<\omega}$, $P_{\times}$ and P_{is} partition the set of places P. A sequence of functions σ is a *ρ-pumping sequence enabled at a marking M_0* if

1. $M_0 \xrightarrow{\sigma}_\rho M_1$ and $M_1 > M_0$,

2. for all places $p \in P_\times$, some function $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ in σ has $\mathbf{B}[\rho](p) = 1$,
3. each function in σ isolates $P_{is} \cup P_{<\omega}$ from $P_\times \cup P_{is}$ and
4. no function in σ multiplies p for any place $p \in P_{is}$.

Next we develop formalisations needed to show that if there are ρ-pumping sequences, there are short ones. Suppose $W_1 \xrightarrow{\sigma}_t W_2$ and $\overline{\omega(W_1)} = \overline{\omega(W_2)}$. Then each function in σ isolates $\overline{\omega(W_1)}$ from $\omega(W_1)$ (otherwise, we could not have $\overline{\omega(W_1)} = \overline{\omega(W_2)}$). The following proposition establishes a relationship between t-transition systems and ρ-transition systems.

▶ Proposition 18. Let t be a threshold function, $W_1 \xrightarrow{\sigma}_t W_2$ and $\overline{\omega(W_1)} = \overline{\omega(W_2)}$. Let $\rho = (\overline{\omega(W_1)}, P_\times, P_{is})$. If $[W_1]_{\omega \to t} \xrightarrow{\sigma}_\rho M_2$, then $M(p) < t(card(\overline{\omega(W_1)}))$ for every intermediate marking M arising while firing $\sigma[\rho]$ from $[W_1]_{\omega \to t}$ and every $p \in \overline{\omega(W_1)}$.

The following definition of loops will be used only in the proof of Lemma 20.

▶ **Definition 19.** Suppose W_1 is an extended marking such that $\overline{\omega(W_1)} = P_{<\omega}$ and σ is a sequence such that all functions in $\sigma[\rho]$ isolate $P_{<\omega}$ from $P \setminus P_{<\omega}$. Suppose σ can be decomposed as $\sigma = \pi_1\pi\pi_2$ and $W_1 \xrightarrow{\pi_1}_\rho L \xrightarrow{\pi}_\rho L \xrightarrow{\pi_2}_\rho W_2$. The pair (π, L) is a *$P_{<\omega}$-loop* if all extended markings (except the last one) arising while firing $\pi[\rho]$ from L are distinct from one another.

▶ **Lemma 20.** *There exists a constant d such that for any strongly increasing Affine net $\mathcal{N}$ and for every ρ-pumping sequence σ enabled at some marking M_0, there exists a ρ-pumping sequence σ' enabled at M_0' such that $|\sigma'| \leq (2eR)^{dm^3}$, where:*

- *$\rho = (P_{<\omega}, P_\times, P_{is})$ is a triple such that $P_{<\omega}$, $P_\times$ and P_{is} partition the set of places P,*
- *$e = 1 + \max\{M(p) \mid p \in P_{<\omega}$ and M is an intermediate marking occurring while firing σ from $M_0\}$ and*
- *M_0' is any marking such that $M_0' =_{P_{<\omega}} M_0$ and $M_0'(p) \geq R|\sigma'|$ for all $p \in P_{is} \cup P_\times$.*

Proof. Let $M_0 \xrightarrow{\sigma}_\rho M_k$. Removing all $P_{<\omega}$-loops from σ may not result in a ρ-pumping sequence. As in the Rackoff technique, we use the existence of small solutions to linear Diophantine equations to show that a small number of loops can be retained to get a shorter ρ-pumping sequence. Some intermediate steps of this proof deal with sub-sequences of σ that may not be enabled at M_0'. They will however be enabled at the extended marking W_0 where $W_0 =_{P_{<\omega}} M_0$ and $W_0(p) = \omega$ for all $p \in P_\times \cup P_{is}$. The proof is organised into the following steps.

Step 1: We first associate a vector with each sub-sequence of σ to measure the effect of the sub-sequence on $P_\times \cup P_{is}$.

Step 2: Next we remove some $P_{<\omega}$-loops from σ to obtain σ'' such that for every intermediate extended marking W arising while firing $\sigma[\rho]$ from W_0, W also arises while firing $\sigma''[\rho]$ from W_0.

Step 3: The sequence σ'' obtained above is not necessarily a ρ-pumping sequence. With the help of the vectors defined in step 1, we formulate a set of linear Diophantine equations to encode the fact that the effects of σ'' and the $P_{<\omega}$-loops that were removed combine to give the effect of a ρ-pumping sequence. This step uses the fact that in ρ-transition systems, $\mathbf{B}[\rho](p)$ is set to 1 for each place $p \in P_\times$, indicating that p is "crossed" by some function.

Step 4: Then we use the result about the existence of small solutions to linear Diophantine equations to infer that only a small number of $P_{<\omega}$-loops need to be retained to ensure that the essential properties of pumping sequences (as encoded by the Diophantine equations) are satisfied. We use this to construct a sequence σ' that meets the length constraint of the lemma.

Step 5: Finally, we prove that σ' is a ρ-pumping sequence enabled at M_0'.

Step 1 is where this proof differs substantially from the ideas used by Rackoff in [16]. We give details of Step 1 here. The missing details are in the full version.

Step 1: Suppose $\pi \in alph(\sigma)^*$ is a sequence consisting of functions occurring in σ. Suppose W_1 is an extended marking such that $W_1(p) \in \mathbb{N}$ for all $p \in P_{<\omega}$ and $W_1(p') = \omega$ for all $p' \in P_\times \cup P_{is}$. Also suppose that $W_1 \xrightarrow{\pi}_\rho W_r$. We want to measure the effect of π on places in $P_\times \cup P_{is}$ when we replace ω by large enough values in W_1. We define a vector $\Delta[\pi, W_1]$ of integers for this measurement. For a place $p \in P_\times$, all that a function $(\mathbf{A}[\rho], \mathbf{B}[\rho], \mathbf{C})$ can do to p is add 0 or 1 (this is due to the way $\mathbf{A}[\rho]$ and $\mathbf{B}[\rho]$ are defined in Definition 16). So we take $\Delta[\pi, W_1](p)$ to be the sum of all $\mathbf{B}[\rho](p)$ for all functions $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ occurring in π. For a place $p \in P_{is}$, we have to take into account the "interference" from other places. Since from Definition 17, each function in π isolates P_{is} from $P_\times \cup P_{is}$, the only places that can interfere with p are those in $P_{<\omega}$. Let $\pi = f_1 f_2 \cdots f_r$ and $W_1 \xrightarrow{f_1}_\rho W_2 \xrightarrow{f_2}_\rho \cdots \xrightarrow{f_{r-1}}_\rho W_r$. For each i between 1 and $r-1$, let $f_i = (\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i)$. Following is the formal definition of $\Delta[\pi, W_1]$:

$$\Delta[\pi, W_1](p) = \sum_{i=1}^{r-1} \left(\sum_{p' \in P_{<\omega}} \mathbf{A}_i(p, p') W_i(p') + \mathbf{B}_i(p) \right) \text{ for all } p \in P_{is}$$

$$\Delta[\pi, W_1](p) = \sum_{i=1}^{r-1} \mathbf{B}_i[\rho](p) \text{ for all } p \in P_\times$$

Since all functions in π isolate $P_{<\omega}$ from $P_\times \cup P_{is}$, we infer that $\overline{\omega(W_r)} = \overline{\omega(W_1)} = P_{<\omega}$. It is routine to infer the following two facts from the definition of $\Delta[\pi, W_1]$.

- If M_1 is any marking such that $M_1 =_{P_{<\omega}} W_1$ and $M_1 \xrightarrow{\pi}_\rho M_2$, then $M_2(p) - M_1(p) = \Delta[\pi, W_1](p)$ for all $p \in P_\times \cup P_{is}$.
- Suppose $\pi = \pi_1\pi_2\pi_3$, $W_1 \xrightarrow{\pi_1}_\rho W' \xrightarrow{\pi_2}_\rho W' \xrightarrow{\pi_3}_\rho W_r$ and (π_2, W') is a $P_{<\omega}$-loop. Then $\Delta[\pi, W_1] = \Delta[\pi_1\pi_3, W_1] + \Delta[\pi_2, W']$.

The above two facts are sufficient to extend the technique used in [16] to ρ-transition systems. This technique was developed for Petri nets, where the effect of a sequence of functions can be determined from its Parikh image. This is not true in general for strongly increasing Affine nets, but the core idea can be lifted to ρ-transition systems.

Details of Steps 2–4 can be found in the full version. ◀

The following lemma establishes what value is "large enough" at the initial marking to ensure that crossed places are unbounded.

▶ **Lemma 21.** *Let $\rho = (P_{<\omega}, P_\times, P_{is})$ be a triple such that the sets $P_{<\omega}$, $P_\times$ and P_{is} partition the set of places P. Suppose σ is a ρ-pumping sequence enabled at M_0. If $M_0(p) \geq 3R|\sigma| + R + 1$ for all $p \in P_\times \cup P_{is}$, then $M_0 \xrightarrow{\sigma} M_1$ and $M_1 > M_0$.*

▶ **Definition 22.** Let $c = 2d$. The functions $\ell_2, t_2 : \mathbb{N} \to \mathbb{N}$ are as follows:

$$t_2(0) = 0 \qquad \ell_2(0) = (2R)^{cm^3}$$

$$t_2(i+1) = 4R\ell_2(i) + R + 1 \qquad \ell_2(i+1) = (2t_2(i+1)R)^{cm^3}$$

Due to the selection of the constant c above, we have $(2xR)^{cm^3} \geq x^i + (2Rx)^{dm^3}$ for all $x \in \mathbb{N}$ and all $i \in \{0, \ldots, m\}$.

▶ **Definition 23.** A t_2-*pumping pair enabled at* W is a pair (σ_1, σ_2) of sequence of functions satisfying the following conditions.

1. $W \xrightarrow{\sigma_1}_{t_2} W_1 \xrightarrow{\sigma_2}_{t_2} W_2$,
2. $\overline{\omega(W_2)} = \overline{\omega(W_1)}$ and
3. for some partition of $\omega(W_1)$ into $P_\times$ and P_{is}, σ_2 is a ρ-pumping sequence enabled at $[W_1]_{\omega \to t_2}$, where $\rho = (\overline{\omega(W_1)}, P_\times, P_{is})$.

The following two lemmas prove that unboundedness in strongly increasing Affine nets is equivalent to the presence of a short t_2-pumping pair enabled at M_{init}. The ideas of these two lemmas are similar to those of Lemma 10 and Lemma 11 respectively. Lemma 20 is used in Lemma 25.

▶ **Lemma 24.** *If* (σ_1, σ_2) *is a* t_2*-pumping pair enabled at* W*, there is a self-covering sequence* (σ'_1, σ'_2) *enabled at* $[W]_{\omega \to t_2}$*.*

▶ **Lemma 25.** *If a self covering pair is enabled at* M_{init}*, there is a* t_2*-pumping pair* (σ'_1, σ'_2) *enabled at* M_{init} *such that* $|\sigma'_1| + |\sigma'_2| \leq \ell_2(m)$*.*

▶ **Lemma 26.** *Let* $k = 8c$*. Then* $\ell_2(i) \leq (2R)^{k^{i+1} m^{3(i+1)}}$ *for all* $i \in \mathbb{N}$*.*

▶ **Theorem 27.** *For some constant* c_3*, the Boundedness problem for strongly increasing Affine nets is in* NSPACE$(\mathcal{O}(2^{c_3 m \log m}(\log R + \log \|M_{init}\|_\infty)))$*.*

With the size of the input $n = (card(F)(m^2 + m) \log R + m \log \|M_{init}\|_\infty)$, we can infer from the above theorem an upper bound of SPACE$(\mathcal{O}(2^{c_4 n \log n}))$ for the boundedness problem.

6 Conclusions and Perspectives

We proved that coverability and boundedness are in SPACE$(\mathcal{O}(2^{cn \log n}))$ for strongly increasing Affine nets. The main difficulty in adapting Rackoff technique is that one cannot simply ignore places that have large enough values, as transitions may copy values from one place to another. From this result, we may immediately deduce the same result for the termination problem as one can add a new place p_{time} which is incremented by every transition. Then, the system terminates iff it is bounded. A natural question is to identify the properties that could be proved (with Rackoff techniques) to be EXPSPACE-complete for strongly increasing Affine nets. At least two (recent) classes of properties are candidates: the generalized unboundedness properties of Demri [5] and the CTL fragment of Blockelet and Schmitz [1]. As this last logic is proved undecidable for strongly increasing Affine nets, a natural restriction of this logic would be defined. We conjecture that replacing the predicates on the effect of a path by predicates on the Parikh image of the path would put the model checking problem for the logic in EXPSPACE.

As we have limited our study to Affine nets, another question would be to consider not only affine functions, but to find classes of recursive functions (that still forbids resets) for which the Rackoff techniques can still be applied. It is likely that the proof of coverability could be adapted by altering Prop. 6 to take into account the "maximum reduction" that functions can perform. For example, if we allow functions to halve the value in a place, it would suffice to say that the initial value is $2^{\ell_1(i)}$ times higher (of course, this would change the final upper bound obtained). However, the proof of boundedness relies heavily on the fact that a place is either fully copied, or not at all, so how to generalize it is unclear.

Coverability and boundedness for Petri nets with an unique Reset/Transfer/Zero-test extended arc have been recently proved to be decidable [2]. For the Zero-test case, the

complexity of coverability is at least as hard as reachability for Petri Nets, so there is not much hope of applying this technique. We conjecture that it could be applied to the one Reset or Transfer case, even if it would yield an upper bound greater than EXPSPACE.

References

1 M. Blockelet and S. Schmitz. Model checking coverability graphs of vector addition systems. In *Proceedings of MFCS*, volume 6907 of *LNCS*, pages 108–119, 2011.

2 R. Bonnet. The reachability problem for vector addition systems with one zero-test. In *Proceedings of MFCS*, volume 6907 of *LNCS*, pages 145–157, 2011.

3 R. Bonnet, A. Finkel, S. Haddad, and F. Rosa-Velardo. Comparing Petri data nets and timed Petri nets. Research Report LSV-10-23, CNRS, ENS Cachan, France, 2010.

4 R. Bonnet, A. Finkel, and M. Praveen. Extending the Rackoff technique to affine nets. Technical report, LSV, ENS Cachan, 2012. http://www.lsv.ens-cachan.fr/Publis/.

5 S. Demri. On selective unboundedness of VASS. In *Proceedings of INFINITY*, volume 39 of *EPTCS*, pages 1–15, 2010.

6 S. Demri, M. Jurdziński, O. Lachish, and R. Lazić. The covering and boundedness problems for branching vector addition systems. In *Proceedings of FSTTCS*, volume 4 of *LIPIcs*, pages 181–192, 2009. Journal version to appear in Journal of Computer and System Sciences, 2012.

7 C. Dufourd and A. Finkel. Polynomial-time many-one reductions for Petri nets. In *Proceedings of FSTTCS*, volume 1346 of *LNCS*, pages 312–326, 1997.

8 C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proceedings of ICALP*, volume 1443 of *LNCS*, pages 103–115, 1998.

9 A. Finkel, P. McKenzie, and C. Picaronny. A well-structured framework for analysing Petri net extensions. *Information & Computation*, 195(1-2):1–29, 2004.

10 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

11 S. R. Kosaraju. Decidability of reachability in vector addition systems. In *Proceedings of STOC*, pages 267–281, 1982.

12 R. Lazic, T. Newcomb, J. Ouaknine, A. W. Roscoe, and J. Worrell. Nets with tokens which carry data. In *Proceedings of ICATPN*, volume 4546 of *LNCS*, pages 301–320, 2007.

13 J. Leroux. Vector addition system reachability problem: a short self-contained proof. In *Proceedings of POPL*, pages 307–316, 2011.

14 R. J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976.

15 E. W. Mayr. An algorithm for the general Petri net reachability problem. In *Proceedings of STOC*, pages 238–246, 1981.

16 C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

17 K. Reinhardt. Reachability in Petri nets with inhibitor arcs. In *Proceedings of RP*, volume 223 of *ENTCS*, pages 239–264, 2008.

18 F. Rosa-Velardo and D. de Frutos-Escrig. Name creation vs. replication in Petri net systems. *Fundamenta Informaticae*, 88(3):329–356, 2008.

19 Ph. Schnoebelen. Lossy counter machines decidability cheat sheet. In *Proceedings of RP*, volume 6227 of *LNCS*, pages 51–75, 2010.

20 Ph. Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *Proceedings of MFCS*, volume 6281 of *LNCS*, pages 616–628, 2010.

21 R. Valk. Self-modifying nets, a natural extension of Petri nets. In *Proceedings of ICALP*, volume 62 of *LNCS*, pages 464–476, 1978.

Accelerating tree-automatic relations

Anthony Widjaja Lin

Oxford University Department of Computer Science

Abstract

We study the problem of computing the transitive closure of tree-automatic (binary) relations, which are represented by tree automata. Such relations include classes of infinite systems generated by pushdown systems (PDS), ground tree rewrite systems (GTRS), PA-processes, and Turing machines, to name a few. Although this problem is unsolvable in general, we provide a semi-algorithm for the problem and prove completeness guarantee for PDS, GTRS, and PA-processes. The semi-algorithm is an extension of a known semi-algorithm for structure-preserving tree-automatic relations, for which completeness is guaranteed for several interesting parameterized systems over tree topology. Hence, there is a single generic procedure that solves reachability for PDS, GTRS, PA-processes, and several parameterized systems in a *uniform* way. As an application, we provide a single generic semi-algorithm for checking repeated reachability over tree-automatic relations, for which completeness is guaranteed for the aforementioned classes.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases Semi-algorithm, Model Checking, Infinite Systems, Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.313

1 Introduction

Real-world systems are complex and many sources of infinity naturally arise when modeling them formally, e.g., recursions in function calls, data structures (lists, trees, etc.) of unbounded size, numeric variables of unbounded size, multi-threading (unbounded number of threads spawned). Given the modeling power of infinite-state systems, even checking simple properties (e.g reachability) easily becomes undecidable over simple classes of infinite systems (e.g. those represented by counter machines). Despite this, many interesting properties (e.g. safety, liveness, temporal-logic specifications) have been shown to be decidable over many classes of infinite-state systems (e.g. pushdown systems, timed systems, Petri nets, lossy channel systems, ground tree rewrite systems, and process rewrite systems). We refer the reader to [1, 12, 24, 26, 25, 27, 31] for a glimpse of these decidability results.

Another common approach to infinite-state model checking is to start with expressive ("Turing-powerful") formalisms that can capture many complex real world features and develop semi-algorithms for model checking that can solve many practical instances. A popular framework for reasoning about complex infinite-state systems has been proposed under the rubric of *regular model checking* (e.g. see [5, 7, 9, 28]), in which systems are represented by "regular" symbolic representations including finite-state automata (over words, trees, ω-words, etc.) or logical formulas over a decidable theory (e.g. Presburger formulas). Many regular model checking frameworks (differing in expressive power) have been considered in the literature, including (extended) counter systems (e.g. see [9, 7]), rational graphs (e.g. see [18, 5]), word-automatic graphs with length-preserving relations (e.g. see [5, 28]), tree-automatic graphs with structure-preserving relations (e.g. see [4, 14]), tree transducers (e.g. see [14]), and a natural subclass of ω-word automatic graphs (e.g. see [11]).

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 313–324

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

Two commonly considered problems in regular model checking are: (1) given a regular symbolic representation of a set X of configurations, compute a regular symbolic representation of the set $post^*(X)$ (or $pre^*(X)$) of reachable configurations, and (2) compute a regular symbolic representation of the transitive closure $\mathcal{R}^*$ of the transition relation $\mathcal{R}$. The second problem is more general than the first (e.g. see [5, 29]). Although these problems are uncomputable in general, semi-algorithms have been developed which can solve these problems for many interesting practical instances (e.g. see [4, 5, 7, 11, 14, 18, 28]), which include *parameterized systems* (i.e. distributed protocols with *any* number of components with some underlying topology, e.g., the dining philosopher problem), which cannot easily be captured by known decidable subclasses. Such semi-algorithms usually employ certain *acceleration* techniques, which compute the effect of arbitrarily long sequences of transitions.

From a theoretical perspective, an important problem when designing semi-algorithms in regular model checking is undoubtedly the question of convergence and completeness. As has been pointed out in [7], many existing semi-algorithms in regular model checking lack general completeness criteria. For example, it was not known if there is a generic semi-algorithm in regular model checking with *completeness guarantee* (i.e. always terminates and gives correct solution) over the *full* class of pushdown systems, let alone strictly more powerful formalisms like ground tree rewrite systems.

Contributions. We investigate regular model checking under the framework of tree-automatic structures [8]. More precisely, we study the problem of computing the transitive closure of a given *tree-automatic (binary) relation* [8]. Such relations are represented by the standard tree-automata over a product alphabet (extended by a padding symbol) allowing configurations (i.e. trees) to grow unboundedly in some paths in the systems. Tree-automatic relations are expressive, i.e., can model the transition graphs of pushdown systems (PDS), PA-processes (PA), ground tree rewrite systems (GTRS), Petri nets, and even Minsky's counter machines and Turing machines.

Our contributions are as follows. We start by showing that the *bounded local depth acceleration* technique given in [4] for structure-preserving tree-automatic relations (i.e. which only relate trees with the same structure) can be extended to the full class of tree-automatic relations. Roughly speaking, for each k and a tree-automatic relation $\mathcal{R}$, the resulting algorithm computes a tree-automatic relation $\lfloor R \rfloor_k$ such that each pair $(T, T') \in \lfloor R \rfloor_k$ has a witnessing path in $\mathcal{R}$, wherein each node is "modified" at most k times. In the case when $\lfloor R \rfloor_k$ is transitive (which can be effectively checked), we have $\mathcal{R}^* = \lfloor R \rfloor_k$. The semi-algorithm simply tries to find a number k such that $\lfloor R \rfloor_k$ is transitive. Abdulla et al. has given some interesting instances of parameterized systems with tree topology for which the semi-algorithm (for structure-preserving case) is guaranteed to terminate [4]. Our main contribution is to show that the extended semi-algorithm is also guaranteed to terminate for several well-studied classes of infinite systems including PDS, PA-processes, and GTRS. As an application, we combine this with the result from [29] to obtain a semi-algorithm for testing repeated reachability, which is guaranteed to terminate for PDS, PA, and GTRS. Hence, we have a *uniform* solution for (repeated) reachability for PDS, PA, and GTRS.

Discussion and Related Work. There are known specialized algorithms for computing the reachability relations (i.e. transitive closure) for PDS [15], PA-processes [26], and GTRS [19, 16] in polynomial time. Although our semi-algorithm has worse upper bounds on the running time, it is more general since it can solve instances of parameterized systems with tree topology [4]. The purpose of this paper has *never* been to provide more efficient algorithms for PA-processes, GTRS, and PDS. Rather, we only intend to show the possibility of devising a *single* generic semi-algorithm that is guaranteed to terminate for the aforementioned classes

of systems, as well as other systems that cannot be easily captured by known decidable subclasses. We leave it for future work to evaluate how the semi-algorithms perform in practice on PA-processes, PDS, and GTRS, and if better acceleration techniques can be devised for them.

Bouajjani & Touili [14] gave a semi-algorithm for computing the reachability sets in the framework of regular tree model checking, where tree transducers (and structure-preserving tree-automatic relations) are adopted. Among others, their semi-algorithms are guaranteed to compute the reachability sets (but not reachability relations) for process rewrite systems (which subsume PA), GTRS (and its extensions), provided that the relations are well-founded (i.e. have no infinite decreasing chain). Process rewrite systems and GTRS are in general not well-founded, but they showed that PA can be transformed into a well-founded PA while preserving reachability. The class of relations generated by tree transducers is more expressive than tree-automatic relations in general, but is less well-behaved than tree-automatic relations (e.g. see [16] and [8]). In particular, it is open whether we can decide repeated reachability, given a tree transducer that generates $\mathcal{R}^*$ (which is the case for tree-automatic relations [29]). There are also other acceleration techniques in regular tree model checking (e.g. see [3, 6, 13]), but they do not have termination guarantee for PA-processes, PDS, and GTRS.

Using *flat acceleration* techniques [7, 17], semi-algorithms for computing the reachability sets (e.g. represented as Presburger formulas) over extended counter systems can be developed that are guaranteed to solve the subcases of reversal-bounded counter systems, and many interesting subclasses of Petri nets (e.g. 2-dim vector addition systems). A similar result of this form can be found in [10], which provides a semi-algorithm which is guaranteed to compute the reachability sets of timed automata (and more general hybrid systems).

2 Preliminaries

General notations For two given natural numbers $i \leq j$, we define $[i, j] = \{i, i+1, \ldots, j\}$. Define $[k] = [0, k]$. Given a set S, we use S^* to denote the set of all finite sequences of elements from S. The set S^* always includes the empty sequence which we denote by ϵ. Given two sets of words S_1, S_2, we use $S_1 \cdot S_2$ to denote the set $\{v \cdot w : v \in S_1, w \in S_2\}$ of words formed by concatenating words from S_1 with words from S_2. Given two relations $R_1, R_2 \subseteq S \times S$, we define their composition as $R_1 \circ R_2 = \{(s_1, s_3) : (\exists s_2)((s_1, s_2) \in R_1 \wedge (s_2, s_3) \in R_2)\}$.

Transition systems Let ACT be a finite set of *action symbols*. A *transition system* over ACT is a tuple $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a \in \mathsf{ACT}} \rangle$, where S is a set of *configurations*, and $\rightarrow_a \subseteq S \times S$ is a binary relation over S. We use $\rightarrow$ to denote the relation $\left(\bigcup_{a \in \mathsf{ACT}} \rightarrow_a\right)$. The notation $\rightarrow^+$ (resp. $\rightarrow^*$) is used to denote the transitive (resp. transitive-reflexive) closure of $\rightarrow$. We say that a sequence $s_1 \rightarrow \cdots \rightarrow s_n$ is a *path* (or *run*) in $\mathfrak{S}$ (or in $\rightarrow$). Given two paths $\pi_1 : s_1 \rightarrow^* s_2$ and $\pi_2 : s_2 \rightarrow^* s_3$ in $\rightarrow$, we may concatenate them to obtain $\pi_1 \odot \pi_2$ (by gluing together s_2). Given a relation $\rightarrow \subseteq S \times S$ and subsets $S_1, \ldots, S_n \subseteq S$, denote by $\textsc{Rec}_{\rightarrow}(\{S_i\}_{i=1}^n)$ to be the set of elements $s_0 \in S$ for which there exists an infinite path $s_0 \rightarrow s_1 \rightarrow \cdots$ visiting each S_i infinitely often, i.e., such that, for each $i \in [1, n]$, there are infinitely many $j \in \mathbb{N}$ with $s_j \in S_i$.

Trees, automata, and languages A *ranked alphabet* is a nonempty finite set of symbols Σ equipped with an arity function $\mathtt{ar} : \Sigma \rightarrow \mathbb{N}$. A *tree domain* D is a nonempty finite subset of $\mathbb{N}^*$ satisfying (1) *prefix closure*, i.e., if $vi \in D$ with $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $v \in D$, (2) *younger-sibling closure*, i.e., if $vi \in D$ with $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $vj \in D$ for each natural number $j < i$. The elements of D are called *nodes*. Standard terminologies (e.g. parents, children, ancestors, descendants) will be used when referring to elements of a tree domain.

For example, the children of a node $v \in D$ are all nodes in D of the form vi for some $i \in \mathbb{N}$. A *tree* over a ranked alphabet Σ is a pair $T = (D, \lambda)$, where D is a tree domain and the *node-labeling* λ is a function mapping D to Σ such that, for each node $v \in D$, the number of children of v in D equals the arity $\mathtt{ar}(\lambda(v))$ of the node label of v. We use the notation $|T|$ to denote $|D|$. Write $\textsc{Tree}(\Sigma)$ for the set of all trees over Σ. We also use the standard term representations of trees (cf. [16]).

A nondeterministic tree-automaton (NTA) over a ranked alphabet Σ is a tuple $\mathcal{A} = \langle Q, \Delta, F\rangle$, where (i) Q is a finite nonempty set of states, (ii) Δ is a finite set of rules of the form $(q_1, \ldots, q_r) \xrightarrow{a} q$, where $a \in \Sigma$, $r = \mathtt{ar}(a)$, and $q, q_1, \ldots, q_r \in Q$, and (iii) $F \subseteq Q$ is a set of final states. A rule of the form $() \xrightarrow{a} q$ is also written as $\xrightarrow{a} q$. A *run* of $\mathcal{A}$ on a tree $T = (D, \lambda)$ is a mapping ρ from D to Q such that, for each node $v \in D$ (with label $a = \lambda(v)$) with its all children $v_1, \ldots, v_r$, it is the case that $(\rho(v_1), \ldots, \rho(v_r)) \xrightarrow{a} \rho(v)$ is a transition in Δ. For a subset $Q' \subseteq Q$, the run is said to be *accepting at* Q' if $\rho(\epsilon) \in Q'$. It is said to be *accepting* if it is accepting at F. The NTA is said to *accept* T at Q' if it has an run on T that is accepting at Q'. Again, we will omit mention of Q' if $Q' = F$. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is precisely the set of trees which are accepted by $\mathcal{A}$. A language L is said to be *regular* if there exists an NTA accepting L. In the sequel, we use $\|\mathcal{A}\|$ to denote the size of $\mathcal{A}$.

A *context* with *(context) variables* $x_1, \ldots, x_n$ is a tree $T = (D, \lambda)$ over the alphabet $\Sigma \cup \{x_1, \ldots, x_n\}$, where $\Sigma \cap \{x_1, \ldots, x_n\} = \emptyset$ and for each $i = 1, \ldots, n$, it is the case that $\mathtt{ar}(x_i) = 0$ and there exists a unique *context node* u_i with $\lambda(u_i) = x_i$. In the sequel, we will sometimes denote such a context as $T[x_1, \ldots, x_n]$. Intuitively, a context $T[x_1, \ldots, x_n]$ is a tree with n "holes" that can be filled in by trees in $\textsc{Tree}(\Sigma)$. More precisely, given trees $T_1 = (D_1, \lambda_1), \ldots, T_n = (D_n, \lambda_n)$ over Σ, we use the notation $T[T_1, \ldots, T_n]$ to denote the tree (D', λ') obtained by filling each hole x_i by T_i, i.e., $D' = D \cup \bigcup_{i=1}^{n} u_i \cdot D_i$ and $\lambda'(u_i v) = \lambda_i(v)$ for each $i = 1, \ldots, n$ and $v \in D_i$. Given a tree T, if $T = C[t]$ for some context tree $C[x]$ and a tree t, then t is called a *subtree* of T. If u is the context node of C, then we use the notation $T(u)$ to obtain this subtree t. Given an NTA $\mathcal{A} = \langle Q, \Delta, F\rangle$ over Σ and states $\bar{q} = q_1, \ldots, q_n \in Q$, we say that $T[x_1, \ldots, x_n]$ is accepted by $\mathcal{A}$ from $\bar{q}$ (written $T[q_1, \ldots, q_n] \in \mathcal{L}(\mathcal{A})$) if it is *accepted* by the NTA $\mathcal{A}' = \langle Q, \Delta', F\rangle$ over $\Sigma \cup \{x_1, \ldots, x_n\}$, where Δ' is the union of Δ and the set containing each rule of the form $\xrightarrow{x_i} q_i$.

3 Tree-automatic relations

Fix a nonempty ranked alphabet Σ. We reserve a special symbol $\perp$ such that $\perp \notin \Sigma$ with $\mathtt{ar}(\perp) := 0$. Let $\Sigma_\perp = \Sigma \cup \{\perp\}$, and $\overline{\perp} := (\perp, \perp)$. In order to define the notion of tree-automatic relations, we will need to first define the convolution operator $\otimes$, which maps a pair of trees over Σ into a tree over the "product alphabet" $\boldsymbol{\Sigma}_\perp := (\Sigma_\perp \times \Sigma_\perp) \setminus \overline{\perp}$ containing labels of the form (a, b) with arity $\mathtt{ar}((a, b)) := \max\{\mathtt{ar}(a), \mathtt{ar}(b)\}$. Given two trees $T_1 = (D_1, \lambda_1)$ and $T_2 = (D_2, \lambda_2)$ over Σ, their convolution is the tree $T_1 \otimes T_2 := (D_1 \cup D_2, \lambda)$ over $\boldsymbol{\Sigma}_\perp$ such that $\lambda(v) = (\lambda_1'(v), \lambda_2'(v))$, where λ_i' is the extension of the function λ_i to the domain $D_1 \cup D_2$ such that $\lambda_i'(w) = \perp$ whenever $w \notin D_i$.

Consider a binary relation $\mathcal{R} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$. We say that the relation is *tree-automatic* [8] (also see [16, Chapter 3]) if the language $\{T \otimes T' : (T, T') \in \mathcal{R}\}$ over $\boldsymbol{\Sigma}_\perp$ is regular. In this case, a *presentation* of $\mathcal{R}$ is any NTA recognizing the language. In the sequel, a presentation of tree-automatic relations is also referred to as a *synchronous (tree)-automaton*. The relation $\mathcal{R} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ is said to be *structure-preserving* if it contains only pairs (T, T') of trees with the same tree domains. Therefore, a synchronous automaton presenting a structure-preserving binary relation runs over the alphabet $\boldsymbol{\Sigma} := \Sigma \times \Sigma$ (i.e.

may not take tuples involving $\perp$).

A *tree-automatic transition system* [8] is a transition system of the form $\mathfrak{S} = \langle S, \{\rightarrow_a\}_{a\in\mathsf{ACT}}\rangle$, where for some ranked alphabet Σ: (1) the domain S is the language of some NTA $\mathcal{A}_{Dom}$ over Σ, and (2) for each action $a \in \mathsf{ACT}$, $\rightarrow_a \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ is a tree-automatic relation presented by some NTA $\mathcal{A}_a$. The tuple $\upsilon = \langle \mathcal{A}_{Dom}; \{\mathcal{A}_a\}_{a\in\mathsf{ACT}}\rangle$ is said to be a *presentation* for the transition system $\mathfrak{S}$. Given a first-order formula $\varphi(x_1, \ldots, x_n)$ over the vocabulary $\{\rightarrow_a\}_{a\in\mathsf{ACT}}$, we write $[\![\varphi]\!]_{\mathfrak{S}}$ for the set of interpretations $(T_1, \ldots, T_n) \in \textsc{Tree}(\Sigma)^n$ such that $\mathfrak{S}$ satisfies the formula φ, i.e., $\mathfrak{S} \models \varphi(T_1, \ldots, T_n)$. When $\mathfrak{S}$ is understood, we simply write $[\![\varphi]\!]$. The following are basic results from the theory of (tree)-automatic structures.

▶ Proposition 1 ([8]). Given a tree-automatic transition system $\mathfrak{S}$ presented by the presentation υ and a first-order sentence φ over the vocabulary of $\mathfrak{S}$, checking whether $\mathfrak{S} \models \varphi$ is decidable. In fact, if φ has free variables x_1, x_2, then a synchronous automaton for $[\![\varphi]\!]_{\mathfrak{S}}$ is computable.

When the input formula is existential, model checking is solvable in exponential time [8]. This implies that given an NTA $\mathcal{A}$ presenting the relation $\mathcal{R}$, checking whether $\mathcal{R}$ is transitive can be done in exponential time. This is because non-transitivity can be expressed as $\exists x, y, z(\mathcal{R}(x, y) \wedge \mathcal{R}(y, z) \wedge \neg\mathcal{R}(x, z))$. A simple analysis of the proof of Proposition 1 (e.g. see [29]) also shows that given an automatic transition system $\mathfrak{S}$ presented by the presentation υ and a *fixed* existential positive (i.e. negation-free) first-order formula $\varphi(x_1, x_2)$, we may compute a synchronous automaton for $[\![\varphi]\!]$ in time polynomial in the size $\|\upsilon\|$ of υ.

4 Synchronized automata of finite local depth

In this section, we consider an arbitrary tree-automatic binary relation $\mathcal{R} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ presented by any given synchronous automaton $\mathcal{A} = \langle Q, \Delta, F\rangle$ over the alphabet $\boldsymbol{\Sigma}_\perp$. We shall keep our terminologies as close to [4] as possible.

The *suffix* of a state $q \in Q$ is the set of contexts $T[x]$ accepted by $\mathcal{A}$ from q, i.e., $\mathit{suff}(q) := \{T[x] : T[q] \in \mathcal{L}(\mathcal{A})\}$. For a subset $Q' \subseteq Q$, define $\mathit{suff}(Q') := \bigcup_{q\in Q'} \mathit{suff}(q)$. A context $T[x] = (D, \tau)$ over the alphabet $\boldsymbol{\Sigma}_\perp$, with the unique node u such that $\tau(u) = x$, is said to be *copying* if for each $v \in D \setminus \{u\}$ it is the case that $\tau(v) = (a, a)$ for some $a \in \Sigma$. The state q is said to be *idempotent* if $\mathit{suff}(q)$ contains only copying contexts, and that, for each transition $(q_1, \ldots, q_m) \xrightarrow{(a,b)} q$, we have $a, b \neq \perp$. [The latter restriction is not imposed in the definition of copying contexts in [4], but is a necessary technical restriction.]

The *prefix* $\mathit{pref}(q)$ of a state $q \in Q$ is defined to be the set of trees accepted by $\mathcal{A}$ at $\{q\}$. A tree $T = (D, \tau)$ over $\boldsymbol{\Sigma}_\perp$ is said to be *copying* if each $v \in D$ satisfies $\tau(v) = (a, a)$ for some $a \in \Sigma$. The state q is said to be a *copying (prefix) state* if $\mathit{pref}(q)$ contains only copying trees.

Local depth Let $Q' \subseteq Q$. Consider a run $\pi := T_0 = (D_0, \tau_0), \ldots, T_m = (D_m, \tau_m)$ through $\mathcal{R}$ such that $T_i \otimes T_{i+1} \in \mathcal{L}(\mathcal{A})$ with a witnessing run ρ_i of $\mathcal{A}$, for each $i \in [m-1]$. Let D denote the set $\bigcup_{i=0}^{m} D_i$, and let $\bar\rho$ denote the sequence $\rho_0, \ldots, \rho_{m-1}$. The *$Q'$-local depth of a node $v \in D$ in π with respect to the runs $\bar\rho$* is defined to be the number of indices $i \in [m-1]$ such that $\rho_i(v) \in Q'$. Intuitively, it is the number of times v is "touched" by Q' in the runs $\bar\rho$. The *Q'-local depth of π with respect to $\bar\rho$* is the maximum of Q'-local depths of nodes $v \in D$ in π wrt $\bar\rho$. The *Q'-local depth of the run π* is the minimum of Q'-local depths of π with respect to *some* witnessing runs of $\mathcal{A}$. The *Q'-local depth of a pair $(T, T') \in \mathcal{R}^*$ of trees (with respect to $\mathcal{A}$)* is the minimum of Q'-local depths of runs $\pi = T_0, \ldots, T_m$ through $\mathcal{R}$,

where $T_0 = T$ and $T_m = T'$. The *Q'-local depth of the relation $\mathcal{R}$ (with respect to $\mathcal{A}$)* is the *supremum* over all Q'-local depths of pairs $(T, T') \in \mathcal{R}^*$. [To understand this last concept, an analogy to the notion of "diameter" of a graph G (supremum of lengths of paths in G) can be drawn.] Whenever Q' is clear, we shall omit mention of Q' and simply say "local depth".

▶ **Example 1.** Let $\Sigma = \{0, 1, \hat{0}, \hat{1}\}$ with $\mathtt{ar}(0) = \mathtt{ar}(1) = 0$ and $\mathtt{ar}(\hat{0}) = \mathtt{ar}(\hat{1}) = 2$. Define the relation $\mathcal{R} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ containing tuples (T, T'), where $T = C[i]$ $(i = 0, 1)$ and $T' = C[\hat{i}(j, k)]$ for some context tree $C[x]$, and some $j, k \in \{0, 1\}$. It is easy to give an NTA $\mathcal{N} = \langle Q, \Delta, F\rangle$ for this relation, where $Q = \{q, q_{cpy}, q_{idm}\}$ with a copying (resp. idempotent) state q_{cpy} (resp. q_{idm}), with $\{q\}$-local depth 1. For example, for each $i = 0, 1$, add to Δ the following transitions: $\xrightarrow{(i,i)} q_{cpy}$, $\xrightarrow{(\bot,i)} q$, $(q_{cpy}, q_{cpy}) \xrightarrow{(\hat{i},\hat{i})} q_{cpy}$, $(q, q) \xrightarrow{(i,\hat{i})} q_{idm}$, $(q_{cpy}, q_{idm}) \xrightarrow{(\hat{i},\hat{i})} q_{idm}$, and $(q_{idm}, q_{cpy}) \xrightarrow{(\hat{i},\hat{i})} q_{idm}$. The unique final state is q_{idm}.

Bounded local depth accelerations Given a positive integer k, the *k-local depth acceleration* $\lfloor\mathcal{R}\rfloor_{k,Q'}$ of the relation $\mathcal{R}$ with respect to the state-set $Q' \subseteq Q$ is the relation containing all pairs $(T, T') \in \mathcal{R}^*$ of Q'-local depth at most k. Define the identity relation $\mathcal{R}_{id} = \{(T, T) : T \in \textsc{Tree}(\Sigma)\}$. Observe that $\mathcal{R} \cup \mathcal{R}_{id} \subseteq \lfloor\mathcal{R}\rfloor_{k,Q'} \subseteq \mathcal{R}^*$ for each positive integer k. In the case when $\mathcal{R}$ has finite Q'-local depth, we have $\lfloor\mathcal{R}\rfloor_{k,Q'} = \mathcal{R}^*$ for some k. Again, whenever Q' is clear from the context, we will simply write $\lfloor\mathcal{R}\rfloor_k$.

▶ **Theorem 2.** *Given a copying state q_{cpy} and an idempotent state q_{idm} of $\mathcal{A}$, the k-local depth acceleration $\lfloor\mathcal{R}\rfloor_k := \lfloor\mathcal{R}\rfloor_{k,Q\setminus\{q_{cpy},q_{idm}\}}$ of $\mathcal{R}$ is tree-automatic. Furthermore, an NTA presenting $\lfloor\mathcal{R}\rfloor_k$ can be computed in time polynomial in $\|\mathcal{A}\|$ and exponential in k.*

The above theorem is proven by a reduction to the computation of k-local depth acceleration of a structure-preserving relation with *two* special copying states; the latter can be proven by a simple adaptation of the proof for the subcase shown in [4] with *one* special copying state q_{cpy}. [In the technical report of [4], an extension is given with several special copying states but has a different condition from the above theorem.] This reduction itself is achieved by "reserving" enough space in the resulting structure-preserving relation (by means of padding) to simulate computation paths in the original system. Details for the reduction and adaptation of the proof of [4] are given in the full version.

Using Theorem 2, we can design a simple semi-algorithm (call it *bounded local depth acceleration semi-algorithm)* for computing an NTA $\mathcal{A}^*$ for the reachability relation $\mathcal{R}^*$ of the tree-automatic relation $\mathcal{R}$ presented by $\mathcal{A}$:

1. Let $k := 1$;
2. **Repeat**
3. Construct $\mathcal{A}^*$ presenting $\lfloor\mathcal{R}\rfloor_k$; let $k := k + 1$;
4. **Until** the relation presented by $\mathcal{A}^*$ is transitive

Checking transitivity of a tree-automatic relation is done using Proposition 1 (see the related remark). Since $\mathcal{R} \cup \mathcal{R}_{id} \subseteq \lfloor\mathcal{R}\rfloor_k \subseteq \mathcal{R}^*$ for each positive integer k, termination implies that the output is an NTA presenting $\mathcal{R}^*$.

We will see other examples of (presentations of) tree-automatic relations with finite local depth in the subsequent sections. The reader is also referred to [4] for nice examples of parameterized systems modeled by structure-preserving tree-automatic relations.

Checking (generalized) repeated reachability The problem of generalized repeated reachability for tree-automatic relations is defined as follows: given an NTA presenting a tree-automatic relation $\rightarrow \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ and a set $\{\mathcal{N}_i\}_{i=1}^m$ of m NTAs over Σ, compute the set $\textsc{Rec}_{\rightarrow}(\{\mathcal{L}(\mathcal{N}_i)\}_{i=1}^m)$ of trees from which there exists a path visiting each $\mathcal{L}(\mathcal{N}_i)$ infinitely often. As for checking safety, this problem is also undecidable (in fact,

Σ^1_1-complete); see [29]. However, it is known that whenever $\rightarrow^*$ is given as an NTA $\mathcal{A}^*$ as part of the input, the problem becomes decidable:

▶ Proposition 2 ([29, 30]). Given two NTAs $\mathcal{A}$ and $\mathcal{A}^*$ presenting, respectively, a relation $\rightarrow$ and its closure $\rightarrow^*$, and given the set $\{\mathcal{N}\}_{i=1}^m$ of m NTAs over the alphabet Σ, we may compute the set $\textsc{Rec}_{\rightarrow}(\{\mathcal{L}(\mathcal{N}_i)\}_{i=1}^m)$ in time polynomial in $\|\mathcal{A}\| \times \|\mathcal{A}^*\| \times \prod_{i=1}^m \|\mathcal{N}_i\|$.

This proposition (proven in [29]) is a corollary of the result on repeated reachability with $m = 1$, which initially appeared in [30] and was proven using Ramsey theory on infinite graphs. Thus, we may combine this proposition with the bounded local depth acceleration semi-algorithm to obtain a semi-algorithm for generalized repeated reachability, which is guaranteed to terminate for relations $\rightarrow$ of finite local depth. In the sequel, we refer to this semi-algorithm as *bounded local depth repeated reachability semi-algorithm*.

5 PA-processes

PA-processes [27] are a well-known generalization of basic parallel processes (BPP) and pushdown systems with one-state (a.k.a. BPA). They are known to be incomparable to both pushdown systems and Petri nets, and are an important class in the well-known process rewrite systems hierarchy [27].

We follow the presentation of [25] to define PA-processes. Let $\mathcal{V} = \{X, Y, \ldots\}$ be a given finite set whose elements are called *process constants*. A *Process term* over $\mathcal{V}$ is simply a tree in $\textsc{Tree}(\Gamma)$, where $\Gamma = \mathcal{V} \cup \{0, \|, \circ\}$ and $\mathtt{ar}(0) = \mathtt{ar}(X) = 0$, for each $X \in \mathcal{V}$, and $\mathtt{ar}(\|) = \mathtt{ar}(\circ) = 2$. Let $\mathcal{F}_{\mathcal{V}} = \textsc{Tree}(\Gamma)$. Here, 0 denotes a "nil" process, and $\circ$ (resp. $\|$) is the sequential (resp. parallel) composition. Process terms are usually represented using standard term representation of trees where $\|$ and $\circ$ are written in infix notations. A *PA declaration* $\mathcal{P}$ over $\mathcal{V}$ is a set of *PA transition rules* of the form $X \rightarrow t$, where $X \in \mathcal{V}$ and $t \in \mathcal{F}_{\mathcal{V}}$. The PA declaration $\mathcal{P}$ defines a relation $\mathcal{R}_{\mathcal{P}} \subseteq \mathcal{F}_{\mathcal{V}} \times \mathcal{F}_{\mathcal{V}}$, which we will also write as $\rightarrow_{\mathcal{P}}$ (in infix notation), as given by the following inference rules:

$$\frac{t_1 \rightarrow_{\mathcal{P}} t_1'}{t_1 \| t_2 \rightarrow_{\mathcal{P}} t_1' \| t_2} \qquad \frac{t_1 \rightarrow_{\mathcal{P}} t_1'}{t_1 \circ t_2 \rightarrow_{\mathcal{P}} t_1' \circ t_2} \qquad \frac{}{X \rightarrow_{\mathcal{P}} t}\ (X \rightarrow t) \in \mathcal{P}$$

$$\frac{t_2 \rightarrow_{\mathcal{P}} t_2'}{t_1 \| t_2 \rightarrow_{\mathcal{P}} t_1 \| t_2'} \qquad \frac{t_2 \rightarrow_{\mathcal{P}} t_2'}{t_1 \circ t_2 \rightarrow_{\mathcal{P}} t_1 \circ t_2'}\ t_1 \in \text{IsNil}$$

Here $\text{IsNil} = \textsc{Tree}(\{0, \circ, \|\})$ is the set of "terminated" process terms.

Due to the condition $t_1 \in \text{IsNil}$ in the inference rule $\frac{t_2 \rightarrow_{\mathcal{P}} t_2'}{t_1 \circ t_2 \rightarrow_{\mathcal{P}} t_1 \circ t_2'}\ t_1 \in \text{IsNil}$, an NTA presenting $\mathcal{R}_{\mathcal{P}}$ will need two special copying states (one to cater for copying trees in IsNil, and another for copying trees that are not in IsNil), which is prohibited in our framework[1]. In order to capture PA within our framework, we will have to modify the above semantics slightly. Extend the ranked alphabet Σ with a new binary symbol $\circ_R$, i.e., $\Sigma = \Gamma \cup \{\circ_R\}$ with $\mathtt{ar}(\circ_R) = 2$. The PA declaration $\mathcal{P}$ now gives rise to another relation $\mathcal{R}_{\mathcal{P},1} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ (also written $\rightarrow_{\mathcal{P},1}$) obtained from the above inference rules, but replacing the inference rule $\frac{t_2 \rightarrow_{\mathcal{P},1} t_2'}{t_1 \circ t_2 \rightarrow_{\mathcal{P},1} t_1 \circ t_2'}\ t_1 \in \text{IsNil}$ by the following two inference rules: **(R1)** $t_1 \rightarrow_{\mathcal{P},1} t_2$, where $t_1 = (D, \tau_1)$ and $t_2 = (D, \tau_2)$ such that there exists a unique $v \in D$ with $\tau_1(v) = \circ$, $\tau_2(v) = \circ_R$, and $\tau_1(u) = \tau_2(u)$ for each $u \in D \setminus \{v\}$ **(R2)** $\frac{t_2 \rightarrow_{\mathcal{P},1} t_2'}{t_1 \circ_R t_2 \rightarrow_{\mathcal{P},1} t_1 \circ_R t_2'}$.

[1] It is possible to extend our framework to allow several special copying states in a way that subsumes PA, which we refrain from doing for space reasons

Intuitively, trees rooted at a node with label $\circ$ (resp. $\circ_R$) can only modify its left (resp. right) subtrees; the only exception to this is an application of Rule (R1). Note that, once $\circ$ is relabeled to $\circ_R$ by Rule (R1), then it cannot be further relabeled.

We now show that the two semantics are *equivalent* in some precise sense. Let $\text{IsNil}' := \textsc{Tree}(\{0, \circ, \circ_R, \|\})$, i.e., the set IsNil where each $\circ$-labeled node may possibly be relabeled by $\circ_R$. Let $\mathcal{F}'_{\mathcal{V}}$ be the set of trees $T \in \textsc{Tree}(\Sigma)$ such that (i) the left subtree of each $\circ_R$-labeled node is in IsNil', and (ii) the left subtree of each $\circ$-labeled node is *not* in IsNil' (i.e. some process constant must be found at a leaf). Observe that we can easily construct an NTA of size $O(|\Sigma|)$ recognizing $\mathcal{F}'_{\mathcal{V}}$. Let $\mathcal{R}_{\leftarrow} \subseteq \mathcal{F}'_{\mathcal{V}} \times \mathcal{F}_{\mathcal{V}}$ consists of tuples (T, T') where T' is obtained by relabeling each $\circ_R$-labeled node in T by $\circ$. Notice that this is also a bijective function (by virtue of the conditions (i) and (ii) above), and additionally a tree-automatic relation which can be presented by an NTA $\mathcal{A}_{\leftarrow}$ of size $O(|\Sigma|)$.

▶ Proposition 3. The following statements are equivalent for all trees T_1, T'_1, T_2, T'_2 with $(T'_1, T_1), (T'_2, T_2) \in \mathcal{R}_{\leftarrow}$:

1. $(T_1, T_2) \in \mathcal{R}^*_{\mathcal{P}}$
2. $(T'_1, T'_2) \in \mathcal{R}^*_{\mathcal{P},1}$

Hence, given a synchronous automaton $\mathcal{A}^*_{\mathcal{P},1}$ presenting $\mathcal{R}^*_{\mathcal{P},1}$, we may compute a synchronous automaton $\mathcal{A}^*_{\mathcal{P}}$ presenting $\mathcal{R}^*_{\mathcal{P}}$ in time polynomial in $\|\mathcal{A}^*_{\mathcal{P},1}\|$.

The proof can be found in the full version.

The relation $\mathcal{R}_{\mathcal{P},1}$ is tree-automatic and can be presented by the NTA $\mathcal{A}_{\mathcal{P},1} = \langle Q, \Delta, F\rangle$ over $\mathbf{\Sigma}_{\perp}$ defined as follows. Suppose that there are n rules in $\mathcal{P}$, where the ith rule is $X_i \to t_i$ where $t_i = (D_i, \tau_i)$. The set Q includes the states q_{cpy}, q_{idm}, $q_{\circ}$, and a state $q_{i,v}$ for each $i \in [1, n]$ and $v \in D_i$. The set F of final states is $\{q_{idm}, q_{\circ}\} \cup \{q_{i,\epsilon} : i \in [1, n]\}$. For each $(a, a) \in \mathbf{\Sigma}$ with $\mathtt{ar}((a, a)) = k$, we add the transition $(q_1, \ldots, q_k) \xrightarrow{(a,a)} q_{cpy}$ to Δ, where $q_1 = \cdots = q_k = q_{cpy}$. Add the transitions $(q_{cpy}, q_F) \xrightarrow{(\|,\|)} q_{idm}$, $(q_F, q_{cpy}) \xrightarrow{(\|,\|)} q_{idm}$, $(q_{cpy}, q_F) \xrightarrow{(\circ_R,\circ_R)} q_{idm}$, and $(q_F, q_{cpy}) \xrightarrow{(\circ,\circ)} q_{idm}$, for each final state $q_F \in F$, to Δ. Add the transition $(q_{cpy}, q_{cpy}) \xrightarrow{(\circ,\circ_R)} q_{\circ}$ to Δ. For each $i \in [1, n]$ and $\epsilon \neq v \in D_i$ with $\mathtt{ar}(\tau_i(v)) = k$, add the transition $(q_{i,v0}, \ldots, q_{i,v(k-1)}) \xrightarrow{(\perp,\tau_i(v))} q_{i,v}$ to Δ. For each $i \in [1, n]$ with $\mathtt{ar}(\tau_i(\epsilon)) = k$, add the transition $(q_{i,0}, \ldots, q_{i,k-1}) \xrightarrow{(X_i,\tau_i(\epsilon))} q_{i,\epsilon}$ to Δ. It is easy to see that $\mathcal{A}_{\mathcal{P},1}$ presents the relation $\mathcal{R}_{\mathcal{P},1}$ and has size $O(\|\mathcal{P}\|)$. Furthermore, it is easy to see that q_{cpy} (resp. q_{idm}) is a copying (resp. idempotent) state. In the sequel, we shall not distinguish $\mathcal{P}$ from $\mathcal{A}_{\mathcal{P},1}$.

▶ **Theorem 3.** *The local depth of a PA $\mathcal{A}_{\mathcal{P},1}$ is at most $O(\|\mathcal{P}\|)$. Therefore, bounded local depth acceleration and repeated reachability semi-algorithms terminate on the class of PA.*

The proof idea is as follows. Suppose $T_1 \to^*_{\mathcal{P},1} T_2$. If a rule $X \to t$ with $|t| > 1$ is applied to an X-labeled node v along a witnessing path, the label of v will next be in $\{\|, \circ, \circ_R\}$. So, the only rule that may modify v in the rest of the path is (R1), which can only be applied at most once at any node. The only problematic case is when we apply rule of the form $X \to Y$, where $X, Y \in \mathcal{V}$. To account for this case, observe that once such rules are applied for more than $|\mathcal{V}|$ times at node v, we have detected redundant applications of PA rules and may remove them to ensure that such rules are applied at most $|\mathcal{V}|$ times at node v. The full proof is given in the full version.

6 Ground tree rewrite systems

Ground-tree rewrite systems (e.g. [24]) are an extension of prefix-rewrite systems (equivalently, pushdown systems), where rewrite rules are given as pairs of trees over some ranked alphabet

and they rewrite subtrees (instead of word prefixes). They are incomparable to PA-processes up to strong bisimulations, but subsume PA-processes up to branching bisimulations [20]. Although local model checking (e.g. fragments of LTL) of PA-processes can be reduced to the same problem over GTRS in polynomial time [20], the same is not known for global model checking (e.g. computing $pre^*(S)$ or reachability relations).

A *ground-tree rewrite systems (GTRS)* [24] over the ranked alphabet Σ is a set $\mathcal{P}$ of rules of the form $t_1 \to t_2$, where $t_1, t_2 \in \textsc{Tree}(\Sigma)$. A *pushdown system (PDS)* is a GTRS where Σ contains no label a with $\mathtt{ar}(a) > 1$. We write $\mathrm{Dom}(\mathcal{P})$ for the set of trees t on the l.h.s. or r.h.s. of some rule in $\mathcal{P}$. The GTRS $\mathcal{P}$ defines a binary relation $\mathcal{R}_\mathcal{P} \subseteq \textsc{Tree}(\Sigma) \times \textsc{Tree}(\Sigma)$ as follows: given two trees $T_1, T_2 \in \textsc{Tree}(\Sigma)$, we have $(T_1, T_2) \in \mathcal{R}_\mathcal{P}$ iff there exists a context $C[x]$ such that $T_1 = C[t_1]$ and $T_2 = C[t_2]$ for some rule $(t_1 \to t_2) \in \mathcal{P}$. For convenience, we will also denote $(T_1, T_2) \in \mathcal{R}_\mathcal{P}$ by $T_1 \to_\mathcal{P} T_2$.

We now give an obvious presentation of the relation $\mathcal{R}_\mathcal{P}$ as a synchronous automaton $\mathcal{A}_\mathcal{P} = \langle Q, \Delta, F\rangle$ over the alphabet $\mathbf{\Sigma}_\perp$. In the sequel, when understood from the context, we will confuse GTRS and their presentations. Suppose that there are n rules in $\mathcal{P}$, where the ith rule is $r_i : t_i \to t_i'$ with $t_i = (D_i, \tau_i)$ and $t_i' = (D_i', \tau_i')$. We extend the function τ_i (resp. τ_i') to $\mathbb{N}^*$ so that whenever $v \notin D_i$ (resp. $v \notin D_i'$) we have $\tau_i(v) = \perp$ (resp. $\tau_i'(v) = \perp$). Then, for each $i \in [1, n]$ and each node $v \in D_i \cup D_i'$, we add a state $q_{i,v}$ to Q. We also add two states q_{idm} and q_{cpy} to Q. We now define the transition relation Δ. For each $a \in \Sigma$ with $\mathtt{ar}(a) = k$, we add the transition $(q_1, \ldots, q_k) \xrightarrow{(a,a)} q_{cpy}$, where $q_1 = \cdots q_k = q_{cpy}$. Such a transition will be used in the subtrees that are not rewritten by $\mathcal{P}$. Likewise, for each $i \in [1, n]$ and node $v \in D_i \cup D_i'$ with children $v0, \ldots, v(k-1)$ (so $vk \notin D_i \cup D_i'$), add the transition $(q_{i,v0}, \ldots, q_{i,v(k-1)}) \xrightarrow{(\tau_i(v), \tau_i'(v))} q_{i,v}$ to Δ. Such a transition will occur in the subtree that is rewritten by $\mathcal{P}$. Finally, for each $a \in \Sigma$ with arity $\mathtt{ar}(a) = k$, we add each possible transition of the form $(q_1, \ldots, q_k) \xrightarrow{(a,a)} q_{idm}$, where: (1) for a unique index $j \in [1, k]$ we have $q_j = q_{idm}$ or $q_j = q_{i,\epsilon}$ for some $i \in [1, n]$, and (2) for each other index $j' \neq j$, we have $q_{j'} = q_{cpy}$. The set F of final states consists of q_{idm} and $q_{i,\epsilon}$ for each $i \in [1, n]$. Note that $\|\mathcal{A}_\mathcal{P}\| = O(\|\mathcal{P}\|)$. Observe that q_{cpy} is a copying state, and q_{idm} an idempotent state. Moreover, it is easy to see that $\mathcal{A}_\mathcal{P}$ presents the relation $\mathcal{R}_\mathcal{P}$.

▶ **Theorem 4.** *The local depth of a GTRS $\mathcal{A}_\mathcal{P}$ is bounded exponentially by $\|\mathcal{A}_\mathcal{P}\|$. So, bounded local depth acceleration and repeated reachability semi-algorithms terminate on GTRS.*

The proof of this theorem is much more involved than the proof for the case of PA-processes. This is because in GTRS (and PDS) a subtree may be rewritten (expanded and shrunk) ad infinitum, unlike for PA-processes where applying any rewrite rule of the form $X \to t$ (with t has more than one node) at a node v guarantees that v can only be touched once more with rule (R1). The proof idea for the theorem is as follows. We will first prove a path factorization lemma for GTRS, which allows us to consider "simpler" paths. Intuitively, simpler paths are paths that visit $\mathrm{Dom}(\mathcal{P})$, i.e., of the form $\pi : T \to^* t \to^* T'$, where $t \in \mathrm{Dom}(\mathcal{P})$. We then prove upper bounds on the local depths of such paths, which will transfer to local depths for $\mathcal{P}$. We will spend the rest of this section to sketch the proof of Theorem 4. We first need the following path factorization lemma for GTRS.

▶ **Lemma 5.** *For two arbitrary trees $T_1, T_2 \in \textsc{Tree}(\Sigma)$, it is the case that $T_1 \to_\mathcal{P}^* T_2$ iff there exists a context tree $C[x_1, \ldots, x_n]$ for some $n \in \mathbb{N}$ such that*

1. *$T_1 = C[t_1, \ldots, t_n]$ for some trees $t_1, \ldots, t_n \in \textsc{Tree}(\Sigma)$.*
2. *$T_2 = C[t_1', \ldots, t_n']$ for some trees $t_1', \ldots, t_n' \in \textsc{Tree}(\Sigma)$.*
3. *for each $i \in [1, n]$, there exists a tree $t_i'' \in \mathrm{Dom}(\mathcal{P})$ such that $t_i \to_\mathcal{P}^* t_i'' \to_\mathcal{P}^* t_i'$.*

Notice that the condition on the r.h.s. of Lemma 5 allows context trees with no variables (i.e. an element of $\textsc{Tree}(\Sigma)$), which accounts for the case when $T_1 \to_{\mathcal{P}}^* T_2$ within zero step (i.e. $T_1 = T_2$). We give the proof of Lemma 5 in the full version.

Let us now prove Theorem 4. Consider any path $\sigma : T_1 \to^* T_2$ in $\mathcal{R}_{\mathcal{P}}$. Let $\mathcal{A}_{cpy}$ denote the NTA obtained from $\mathcal{A}_{\mathcal{P}}$ by removing all transitions in $\mathcal{A}_{\mathcal{P}}$ of the form $(q_1, \ldots, q_k) \xrightarrow{(a,b)} q$ with $q, q_1, q_2, \ldots, q_k \in Q \setminus \{q_{cpy}, q_{idm}\}$. That is, we remove the transitions which rewrite subtrees. Suppose now that $T_1 \to_{\mathcal{P}}^* T_2$. By Lemma 5, there exists a context tree $C[x_1, \ldots, x_n] = (D_C, \tau_C)$, along with trees $t_i, t_i', t_i'' \in \textsc{Tree}(\Sigma)$ (for each $i \in [1, n]$), satisfying the three stated conditions. Let $u_1, \ldots, u_n$ denote the context nodes of C. Let $\mathcal{C}$ denote the context tree (D_C, τ_C') obtained from C by replacing each label a in non-context nodes by (a, a) and leave variables in context nodes as they are, i.e., $\mathcal{C}$ is over the alphabet $\Sigma_{id} := \{(a, a) : a \in \Sigma\} \cup \{x_1, \ldots, x_n\}$ with $\mathtt{ar}((a, a)) = \mathtt{ar}(a)$ such that, if v is not a context node in C, then $\tau_C'(v) = (\tau_C(v), \tau_C(v))$; otherwise, $\tau_C'(v) = \tau_C(v)$. Then, for each $i \in [1, n]$, we see that $\mathcal{C}[q_1, \ldots, q_n] \in \mathcal{L}(\mathcal{A}_{cpy})$, for every $\bar{q}$ satisfying $q_i \in F$ and $q_j = q_{cpy}$ $(j \neq i)$. We now need the following lemma.

▶ **Lemma 6.** *Given two trees $t \in Dom(\mathcal{P})$ and $T \in \textsc{Tree}(\Sigma)$, if $t \to_{\mathcal{P}}^* T$, then there exists a witnessing run $\pi : t \to_{\mathcal{P}}^* T$ of bounded local depth that is exponentially bounded by $\|\mathcal{P}\|$.*

By considering the new GTRS $\mathcal{P}^{-1}$ obtained by swapping the l.h.s. with the r.h.s. of each rule in $\mathcal{P}$, this lemma also implies that $T \to_{\mathcal{P}}^* t$ has a witnessing run of local depth that is exponentially bounded by $\|\mathcal{P}\|$. By condition (3) in Lemma 5, this lemma gives a path $\pi : T_1 \to_{\mathcal{P}}^* T_2$ in $\mathcal{R}_{\mathcal{P}}$ of local depth that is exponentially bounded by $\|\mathcal{P}\|$, which implies Theorem 4.

It remains to prove Lemma 6. To this end, we will need two ingredients. The first ingredient (Lemma 7) is an upper bound on local depth of a *given* tuple (T_1, T_2) with $T_1 \to_{\mathcal{P}}^* T_2$, which will be given by bounds on the length of the shortest witnessing paths. The second ingredient (Lemma 8) is a lemma for using the first ingredient to derive the maximum of local depths of *all* tuples (T_1, T_2) with $T_1 \to_{\mathcal{P}}^* T_2$.

▶ **Lemma 7.** *Given two trees $T_1, T_2 \in \textsc{Tree}(\Sigma)$, if $T_1 \to_{\mathcal{P}}^* T_2$, then T_1 can reach T_2 within at most k steps, where k is exponentially bounded by $\|\mathcal{P}\| + \|T_1\| + \|T_2\|$.*

▶ **Lemma 8.** *Let $T_1 = (D_1, \tau_1), T_2 = (D_2, \tau_2) \in \textsc{Tree}(\Sigma)$. Then, $T_1 \to_{\mathcal{P}}^* T_2$ implies that there exists a context tree $C[x_1, \ldots, x_n] = (D', \tau')$, for some $n \in \mathbb{N}$, with context nodes $u_1, \ldots, u_n$ such that*

1. *$\{u_1, \ldots, u_n\} \subseteq D_2$, $\{u_1, \ldots, u_n\} \cap D_1 = \emptyset$ and $D' \setminus \{u_1, \ldots, u_n\} = D_1 \cap D_2$, i.e., each u_i is a node of D_2 of the form vj for some leaf node v in D_1 and some $j \in \mathbb{N}$.*
2. *for some trees $t_1, \ldots, t_n \in \textsc{Tree}(\Sigma)$ with $|t_i| \leq \|\mathcal{P}\|$ for each $i \in [1, n]$, we have $T_1 \to_{\mathcal{P}}^* C[t_1, \ldots, t_n]$,*
3. *$T_2 = C[t_1', \ldots, t_n']$ for some trees $t_1', \ldots, t_n' \in \textsc{Tree}(\Sigma)$ with $t_i \to_{\mathcal{P}}^* t_i'$ for each $i \in [1, n]$,*

Lemma 7 is an immediate corollary of [23, Theorem 1 and Lemma 2]. Intuitively, Lemma 8 means that $T_1 \to_{\mathcal{P}}^* T_2$ can go via a tree $C[t_1, \ldots, t_n]$, which is not much bigger than T_1, such that $T_2 = C[t_1', \ldots, t_n']$ and $t_i \to_{\mathcal{P}}^* t_i'$ for each $i = 1, \ldots, n$. That is, if we assume that T_1 is small, the intermediate configuration $C[t_1, \ldots, t_n]$ is also small, and we can use Lemma 7 on the path from T_1 to $C[t_1, \ldots, t_n]$ and apply the same reasoning on the path from t_i to t_i', for each $i \in [1, n]$, since each t_i is small.

Proof of Lemma 8. Let us take the unique context tree $C[x_1, \ldots, x_n] = (D', \tau')$ as defined by Condition 1 of the statement of the lemma (note that this context tree depends only on

T_1 and T_2). Let $t'_1, \ldots, t'_n \in \textsc{Tree}(\Sigma)$ be the unique trees such that $T_2 = C[t'_1, \ldots, t'_n]$. It suffices to show that Condition 2 and 3 are satisfied. Let $\pi : G_0 \to_{\mathcal{P}} \cdots \to_{\mathcal{P}} G_m$ be a path such that $G_0 = T_1$ and $G_m = T_2$. Then, for each $i \in [1, m]$ there exists a context tree $C_i[x]$ with context node $v_i \in \mathbb{N}^*$ and a rule $\alpha_i \to \alpha'_i$ in $\mathcal{P}$ such that $G_{i-1} = C_i[\alpha]$ and $G_i = C_i[\alpha'_i]$. For each $i \in [1, n]$, let n_i be the maximum index $j \in [1, m]$ such that $v_j \prec u_i$ (i.e. v_j is an ancestor of u_i excluding u_i), which must exist since $u_i \notin D_1$. Intuitively, n_i denotes the last point in the path π which modifies ancestors of u_i (excluding itself). This means also that u_i is not a node of G_{n_i-1}, but is a node of G_{n_i} (introduced by the rule $\alpha_{n_i} \to \alpha'_{n_i}$ of $\mathcal{P}$). Let t_i denote the subtree of G_{n_i+1} rooted at the node u_i. We have $|t_i| \leq |\alpha'_i| \leq \|\mathcal{P}\|$. It is also easy to see that $t_i \to^*_{\mathcal{P}} t'_i$. A witnessing path is the sequence $G_{n_i+1}(u_i), \ldots, G_m(u_i)$ (removing duplicates). [Recall that $T(u)$ denotes the subtree of T rooted at u.] Furthermore, the path witnessing $T_1 \to^*_{\mathcal{P}} C[t_1, \ldots, t_n]$ can be obtained from π by removing each application of rules that operate on descendants of u_i after G_{n_i+1} (for each $i \in [1, n]$). ◀

To prove Lemma 6, we use Lemma 8 starting with $T_1 = t \in \text{Dom}(\mathcal{P})$. Reaching $C[t_1, \ldots, t_n]$ requires a path of length exponential in $\|\mathcal{P}\|$ by Lemma 7. We now apply the same reasoning again to $t_i \to^*_{\mathcal{P}} t'_i$, for each $i \in [1, n]$, and so build the path from $C[t_1, \ldots, t_n]$ to $C[t'_1, \ldots, t'_n]$ in this fashion. Since each application of this reasoning can only modify descendants of the root of T_1 of at most exponential distance, the constructed path witnessing $t \to^*_{\mathcal{P}} T$ has local depth that is exponential in $\|\mathcal{P}\|$, which completes the proof of Lemma 6. A more detailed argument is given in the full version.

▶ Remark. Even for PDS, an exponential upper bound on the local depth of $\mathcal{P}$ is tight. It is well-known that the shortest path from a configuration C to another configuration C' in PDS can be exponential in the size of the PDS $\mathcal{P}$ (more precisely, in the number of states). A witnessing PDS exhibiting this lower bound (e.g. see [21]) simply represents a number with k bits in the stack in binary, and counts from 0 to $2^k - 1$. Since the least significant bit has to be toggled $2^k - 1$ times, the local depth of the PDS also has $2^k - 1$ as the lower bound.

7 Future work

We mention two possible future avenues: (1) Can other interesting classes of infinite systems (e.g. PAD-processes [27] and order-2 collapsible pushdown automata [22]) be captured within bounded local depth acceleration framework? (2) Improve the framework of bounded local depth acceleration techniques so that the semi-algorithm has faster termination guarantee for PA-processes, PDS, and GTRS. One technique that might also help the latter is the simulation-based antichain technique for language inclusion proposed in [2].

Acknowledgements Many thanks to P. Abdulla for answering questions about [4], M. Hague and anonymous reviewers for their comments, and EPSRC (H026878) for the support.

References

1 P. A. Abdulla, K. Cerans, B. Jonsson, and Y. K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
2 P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *TACAS*, pages 158–174, 2010.
3 P. A. Abdulla, N. B. Henda, G. Delzanno, F. Haziza, and A. Rezine. Parameterized tree systems. In *FORTE*, pages 69–83, 2008.
4 P. A. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular tree model checking. In *CAV*, pages 555–568, 2002.

5 P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR*, pages 35–48, 2004.

6 P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Tree regular model checking: A simulation-based approach. *J. Log. Algebr. Program.*, 69(1-2):93–121, 2006.

7 S. Bardin, A. Finkel, J. Leroux, and Ph. Schnoebelen. Flat acceleration in symbolic model checking. In *ATVA*, pages 474–488, 2005.

8 A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.

9 B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces.* PhD thesis, Université de Liège, 1999.

10 B. Boigelot and F. Herbreteau. The power of hybrid acceleration. In *CAV*, pages 438–451, 2006.

11 B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large (extended abstract). In *CAV*, pages 223–235, 2003.

12 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.

13 A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular (tree) model checking. *STTT*, 14(2):167–191, 2012.

14 A. Bouajjani and T. Touili. Widening techniques for regular tree model checking. To appear in *Int J Softw Tools Technol Transfer*, (Published online: Aug. 2011).

15 D. Caucal. On the regular structure of prefix rewriting. In *CAAP*, pages 87–102, 1990.

16 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007.

17 H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV*, pages 268–279, 1998.

18 D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. *J. Log. Algebr. Program.*, 52-53:109–127, 2002.

19 M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *LICS*, pages 242–248, 1990.

20 S. Göller and A. W. Lin. Refining the process rewrite systems hierarchy via ground tree rewrite systems. In *CONCUR*, pages 543–558, 2011.

21 M. Hague and A. W. Lin. Model checking recursive programs with numeric data types. In *CAV*, pages 743–759, 2011.

22 A. Kartzow. Collapsible pushdown graphs of level 2 are tree-automatic. In *STACS*, pages 501–512, 2010.

23 A. W. Lin. Weakly synchronized ground tree rewriting. In *MFCS*, pages 630–642, 2012.

24 C. Löding. *Infinite Graphs Generated by Tree Rewriting.* PhD thesis, RWTH Aachen, 2003.

25 D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theor. Comput. Sci.*, 274(1-2):89–115, 2002.

26 D. Lugiez and Ph. Schnoebelen. Decidable first-order transition logics for PA-processes. *Inf. Comput.*, 203(1):75–113, 2005.

27 R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, TU-Munich, 1998.

28 M. Nilsson. *Regular Model Checking.* PhD thesis, Uppsala Universitet, 2005.

29 A. W. To. *Model Checking Infinite-State Systems: Generic and Specific Approaches.* PhD thesis, LFCS, School of Informatics, University of Edinburgh, 2010.

30 A. W. To and Leonid Libkin. Recurrent reachability analysis in regular model checking. In *LPAR*, pages 198–213, 2008.

31 I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

k-delivery traveling salesman problem on tree networks*

Binay Bhattacharya[1] and Yuzhuang Hu[2]

1 **School of Computing Science, Simon Fraser University, Burnaby, Canada, V5A 1S6**
binay@cs.sfu.ca

2 **School of Computing Science, Simon Fraser University, Burnaby, Canada, V5A 1S6**
huyuzhuang@gmail.com

Abstract

In this paper we study the k-delivery traveling salesman problem (TSP) on trees, a variant of the non-preemptive capacitated vehicle routing problem with pickups and deliveries. We are given n pickup locations and n delivery locations on trees, with exactly one item at each pickup location. The k-delivery TSP is to find a minimum length tour by a vehicle of finite capacity k to pick up and deliver exactly one item to each delivery location. We show that an optimal solution for the k-delivery TSP on paths can be found that allows succinct representations of the routes. By exploring the symmetry inherent in the k-delivery TSP, we design a $\frac{5}{3}$-approximation algorithm for the k-delivery TSP on trees of arbitrary heights. The ratio can be improved to $(\frac{3}{2} - \frac{1}{2k})$ for the problem on trees of height 2. The developed algorithms are based on the following observation: under certain conditions, it makes sense for a non-empty vehicle to turn around and pick up additional loads.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases k-delivery traveling salesman problem approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.325

1 Introduction

In the capacitated vehicle routing problem with pickups and deliveries (CVRPPD), we are given an undirected complete graph $G=(V, E)$, with edge costs satisfying the triangle inequality, and each vertex is associated with a pickup or a delivery load. A vehicle, or a fleet of vehicles with finite capacity k, traverses the edges of G and serves all the pickup and delivery nodes. During the traversal, the vehicle should never exceed its capacity. The objective is to find a minimum length tour, which starts at the depot and traverses all the nodes of G while satisfying the pickup/delivery demands of the nodes. CVRPPD can capture many real-world transportation and distribution problems. It is a generalization of the traveling salesman problem, and is therefore NP-hard. The problem can further be generalized based on whether preemption of loads of vehicles is allowed. The preemptive version allows the vehicle to temporarily unload the items at an intermediate point of the network which will later be picked up for the delivery. In this note we consider the case where the preemption of loads is not allowed. It is not difficult to see that any solution to

* This work was partially supported by MITACS and NSERC.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 325–336

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

CVRPPD without preemption is an upper bound on the optimal solution where preemption is allowed. A survey of these problems can be found in [5, 6].

In this paper, we focus on a variant of CVRP with pickups and deliveries, called the k-delivery traveling salesman problem.

1.1 k-Delivery traveling salesman problem (k-delivery TSP)

Here the vertex set $V = V_p \cup V_d$ is partitioned into a set V_p of *pickup vertices* and a set V_d of *delivery vertices*. Each vertex of V_p provides an item, and each vertex of V_d requires one item. A vehicle with capacity k starts from a depot and gathers items from the pickup vertices, and delivers them to the delivery vertices before returning to the depot. All the items are identical, therefore an item picked up from a vertex in V_p can be delivered to any vertex in V_d. We assume here $|V_p| = |V_d|$, and the capacity $k < |V_p|$. The objective of the k-delivery TSP is to determine a minimum cost tour of the nodes of G subject to the vehicle capacity constraint.

The k-delivery TSP is also called the capacitated pickup and delivery TSP (CPDTSP) in [8]. Lim et al. [8] showed that the problem can be solved optimally in time $O(n^2/min(k, n))$ on paths, where n is the number of vertices of G. The authors proved in the same paper that the k-delivery TSP is NP-hard in the strong sense even for trees with height 2, using a reduction from the 3-partition problem. A 2-approximation algorithm for the k-delivery TSP on trees is later given in [7]. This algorithm follows a rule that the vehicle would continue to pick up (or deliver) items if possible. The best known approximation ratio of 5 for the k-delivery TSP on general graphs is due to Charikar et al. in [4]. The first constant factor approximation algorithm for the problem was proposed by Chalasani et al. in [2]. A restricted version of the problem was considered in [1].

1.2 Our results

Our results described in this paper are summarized as follows:

1. For the k-delivery TSP on paths, a linear time algorithm is proposed to find the optimal solution. This improves the running time $O(n^2/k)$ of the algorithm in [8].
2. For the k-delivery TSP on trees, we improve the approximation ratio to $\frac{5}{3}$. The best known approximation ratio 2 is due to Lim et al. in [7]. This ratio can be improved to $\frac{3}{2} - \frac{1}{2k}$ for trees with height 2.

All the developed algorithms use a *come-back* rule. Under this rule the vehicle would not be allowed to cross a particular edge e if some certain condition is met. According to the come-back rule, the vehicle postpones delivering its load immediately and picks up more items instead for a better solution. As a consequence, on its way to pick up more loads, the vehicle may traverse several edges without delivering any items, even when it has a non-empty load. This is somewhat contrary to our intuition. However, in this paper we show that this strategy is effective for the k-delivery TSP on paths and trees.

2 Lower bounds

We describe a widely used lower bound, called *the flow bound*, for tree networks for the considered problems. Some notations used to describe the lower bound are similar to the one used in [3] and [8].

It is assumed in this paper that the depot node is located at the root of the tree network. The vehicle starts and ends its tour at the depot node. The flow bound for the k-delivery TSP on trees is described below. Denote a subtree rooted at vertex u by T_u and the parent of u by $p(u)$. For each vertex v of G, we associate a label $a(v)$ with it, which is set to 1 if it is a pickup vertex, and -1 if it is a delivery vertex. For trees, we define $g(v) = \sum_{t \in T_v} a(t)$ to be the net number of items of all the vertices in T_v. Then any k-delivery TSP tour must traverse edge $e = (p(v), v)$ at least $2max\{\lceil \frac{|g(v)|}{k} \rceil, 1\}$ times.

Note here that the flow bound described above is a preemptive lower bound. For all the problems we solved, we bound our solutions to the optimum of the *preemptive* version of these problems.

3 Linear time algorithm for the k-delivery TSP on paths

As implied by the flow bound, an optimal solution of the problem on paths may contain $O(n^2/k)$ edges. The algorithm in [8] takes $O(n^2/k)$ time since it constructs the final optimal solution by explicitly enumerating its edge sequences. However, in this paper we show that our optimal schedule has a succinct linear size representation and the optimal schedule can be computed in linear time.

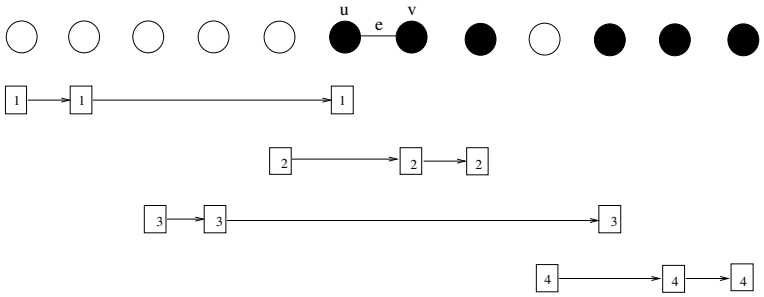


Figure 1 A succinct representation of an optimal solution for the k-delivery TSP on a path where k=2. A white circle represents a pickup vertex and a black circle represents a delivery vertex.

In the succinct representation (see Figure 1), the solution comprises several subroutes, $r_1, r_2, \cdots, r_t$. Each vertex is marked to be picked up or delivered by one such subroute. For each subroute r_i, where $1 \leq i \leq t$, the vehicle travels along the path from the leftmost vertex to the rightmost vertex, which are marked by this subtour. In the meantime, the vehicle services all the vertices marked by the subtour. After reaching the rightmost vertex, the vehicle comes back to the leftmost vertex marked by the next subtour r_{i+1}, and the above process continues. It is clear that such a representation needs only $O(n)$ space.

Let $P =< v_1, v_2, ..., v_n >$ be the path network where the depot is located at v_1. Note that v_1 could be a pickup or a delivery vertex. Define $g(v_i) = \sum_{t=1}^{i} a(v_t)$ to be the net number of items among vertices to the left of v (including v). Using the values $g(v)$ of the nodes v of P, we partition P into path segments P_i with endpoints q_i and q'_i, $i = 1, 2, ..$, where $g(q'_i) = 0$, and $g(w)$ is nonzero for all the remaining vertices w of the segment P_i. The vehicle traverses P_i from the left if $g(q_i) > 0$, otherwise P_i is traversed from the right. Note that any feasible schedule will traverse every edge of the path at least twice. Therefore, without any loss of generality we assume that, except the last vertex, $g(w) > 0$ for all w of P. The main contribution of our algorithm is a strategy called *come-back-path*. The procedure scans the path from left to right and maintains a route number and two stacks. The route number

represents the number of the current subtour, and is used to mark each vertex for a succinct representation. A stack called *vehicle-stack* is used to simulate the behaviour of the vehicle; pushing a vertex to vehicle-stack has the same effect as loading one item from the vertex into the vehicle, and popping a vertex from vehicle-stack means the vehicle delivers one item to the vertex. During the processing, a pickup vertex is put into vehicle-stack if the current vehicle load is less than k, otherwise, it is stored in another stack called *repository-stack*.

The algorithm can be briefly stated as follows:

Procedure come-back-path {
Scan the vertices from left to right.
Step (a) (The vertex is a pickup vertex.) The vertex is pushed into vehicle-stack if the size of the stack is less than k. Otherwise it is pushed into repository-stack.
Step (b) (The vertex is a delivery vertex.) An item from vehicle-stack is popped and delivered to the delivery vertex.
- If the next vertex to be processed is also a delivery vertex, check if the total size of the two stacks is a multiple of k.
- If the condition is true, the action is not to proceed to the next delivery vertex. Instead a new subroute is created, and the vehicle goes back to pick up more loads. This is accomplished by transferring loads from repository-stack to fill up vehicle-stack. The size of vehicle-stack is now k.

}

We define $b_1(t)$ and $b_2(t)$ to be the number of items in vehicle-stack and repository-stack respectively at time step t. By our assumption, the vehicle is moving from left to right. Let the vehicle reach a delivery vertex u at time step t'. The come-back-path strategy applies when the next vertex v is also a delivery vertex. The triggering condition for the come-back-path strategy here is whether $(b_1(t') + b_2(t')) \bmod k = 0$. In other words, the vehicle is allowed to cross edge $e = (u, v)$ from delivery vertex u to delivery vertex v only if $(b_1(t') + b_2(t')) \bmod k \neq 0$. Otherwise, the current subtour is terminated and the vehicle comes back to pick up the topmost $(b_2(t') \bmod k)$ items in repository-stack. To ease the analysis, we assume that the vehicle picks up these items on its way back from u (when the vehicle is moving leftwards). This is achieved by increasing the route number by 1 and transferring the topmost $b_2(t') \bmod k$ items of repository-stack to vehicle-stack.

The come-back-path strategy guarantees that the solution produced by this procedure satisfies the flow bound for the k-delivery TSP. This is stated in the following lemma.

▶ **Lemma 1.** *For an edge e of the path segment, let the subtours found by procedure come-back-path passing through e be $r_1, r_2, \cdots, r_m$. The vehicle crosses e from left to right in each of the routes $r_3, \cdots, r_m$, with exactly k items. Moreover, the vehicle carries the rest (more than k) items in r_1 and r_2.*

Proof. Omitted. ◀

The following theorem is implied by the above lemma.

▶ **Theorem 2.** *For an edge $e = (u, v)$ of the path segment, the tour determined by procedure come-back-path crosses e exactly $2\lceil \frac{|g(u)|}{k} \rceil$ times which is the* flow bound *of e.*

4 A $\frac{5}{3}$-approximation algorithm for the k-delivery TSP on trees

Wang et al. [8] showed that the problem is NP-complete in the strong sense even on trees of height 2. In this section we present a $\frac{5}{3}$-approximation algorithm called *half-load* for the

k-delivery TSP on trees of arbitrary heights. The half-load algorithm is also based on the come-back rule and it achieves a $(\frac{3}{2} - \frac{1}{2k})$-approximation for the k-delivery TSP on trees of height 2. Our initial focus is on height 2 tree networks. The methodology will then be generalized to accommodate arbitrary height tree networks.

Let $T = (V, E)$ denote the tree network containing n vertices. Let V_p and V_d be the set of pickup and delivery vertices of T. We are assuming that the tree network is rooted and the depot location is at the root. We call a subtree T_u positive (negative) if it contains more (less) vertices from V_p than from V_d. An edge $e = (p(u), u)$ or the vertex u is positive (negative) if subtree T_u is positive (negative). Here $p(u)$ is the parent node of the non-root node u. For an edge $e = (p(u), u)$, we say some vertices/items are picked up from (delivered to) e or u, if we pick up (deliver) these vertices/items from (to) the subtree T_u; we say e has a pickup (delivery) load of $g(e)$ if exactly $g(e)$ (which is equal to $|g(u)|$) net number of items need to be picked (delivered) through e. We say the vehicle visits (crosses, traverses) e if the vehicle moves from $p(u)$ to u. Finally we write $LB_e = \lceil \frac{g(e)}{k} \rceil$. Any optimal solution to the k-delivery TSP will traverse edge e at least $2LB_e$ times.

4.1 Exploring the symmetry of the k-delivery TSP

Our improvements for the k-delivery TSP on trees explore the symmetry inherent in the problem. In the k-delivery TSP, the underlying graph has only two types of vertices, it's not difficult to see that if we flip the type of each vertex, to get a new graph, say $\overline{G}$, then the k-delivery TSP has the following property.

▶ **Lemma 3.** *Any feasible solution of the k-delivery TSP on $\overline{G}$ can be converted to a feasible solution of G with the same edge cost, by reversing its edge directions of the routes.*

This property allows us to design approximation algorithms for the k-delivery TSP on trees in the following way. We partition edge set E of the graph into two sets $D(G)$ and $E - D(G)$ based on some definition D. Assume $D(\overline{G}) = E - D(G)$. We have

▶ **Lemma 4.** *Let A be an algorithm that produces tours where each edge in $D(G)$ and $E - D(G)$ is traversed at most α and β times LB_e respectively. Then there exists a solution generated by A with cost no more than $\sum_e \frac{\alpha+\beta}{2} LB_e$.*

Proof. Let τ_1 be a tour after applying algorithm A on G. Flip the type of each vertex and obtain a second tour τ_2 by running the same algorithm on $\overline{G}$. Since $D(G) = E - D(\overline{G})$, each edge e would be crossed at most $(\alpha + \beta) \cdot LB_e$ times in the two tours. The tour with the smaller cost is the desired solution. ◀

Assume algorithm A stated in Lemma 4 is particularly "good" to $D(G)$, in the sense that $\alpha < \beta$. Then Lemma 4 shows that the approximation ratio of A can be further reduced to $\frac{\alpha+\beta}{2}$. In the proposed half-load algorithm, $D(G)$ is defined to be the set of positive edges of G. For the k-delivery TSP on trees of height 2, this algorithm achieves $\alpha = 1$ and $\beta = 2$.

4.2 Two phases of the half-load algorithm

The half-load algorithm being proposed here for the k-delivery TSP on trees uses a strategy called *come-back-tree*. The algorithm contains two phases, the planning phase and the route generating phase. In the planning phase, the original graph G is transformed to a multi-graph G' as follows. In G' each positive tree edge $e = (p(u), u)$ is split into several pseudo edges with total load equal to $g(e)$. Intuitively a pseudo edge $e = (p(u), u)$ records the number

of items that to be collected from T_u during one visit from $p(u)$ to u. Examples of the transformation are given in Figures 2 and 3.

A pseudo edge is treated as a scheduling unit in the second route generating phase. Each time when the vehicle crosses a positive tree edge $e = (p(u), u)$ to service nodes in $T(u)$, we make sure that the vehicle picks up the full load assigned to a distinct pseudo edge between $p(u)$ and u. Thus under the half-load algorithm the number of times e is crossed, is determined by the number of pseudo edges incident to $p(u)$ and u.

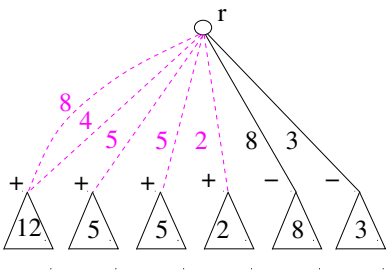


Figure 2 An example of the pseudo edges on a tree of height 2. k=8.

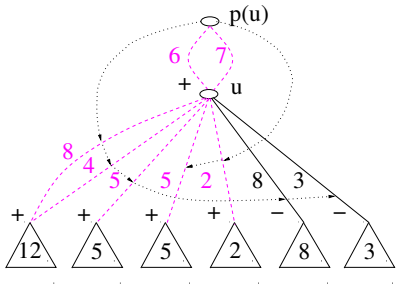


Figure 3 An example of building pseudo edges for a vertex u. k=8.

In the example in Figure 2, the capacity k is set to 8. A pseudo edge of the transformed tree is represented by a dashed line, and a subtree is represented by a triangle with a number showing the net number of items inside this subtree. We assume in G' all the pickup and delivery demands are on the leaves. This can be done easily by adding a pseudo vertex u' for each non-leaf vertex u of G, and attaching an edge between u and u' with zero cost. Since the first positive tree edge in Figure 2 has a load of more than k, it is split into two edges with loads 8 and 4 respectively. Other tree edges do not split, since they are either negative or their loads are less than k.

Note that in the planning phase issues related to route generation, such as the route feasibility constraints, are not considered. The final schedule is computed in the second phase by nested recursive calls of several procedures. A brief description of the second phase is as follows. First a procedure called *come-back-tree* that adopts the come-back rule is applied on the first level of the tree to get the servicing sequence of the involved pseudo and negative tree edges. During the process, two recursive procedures, *pickup* and *deliver*, are called to service the pseudo and negative tree edges respectively. Besides some special processing for picking up and delivering items, procedure come-back-tree is invoked in both procedures on the second level of the tree. This process is continued until a leaf node is met. In all the procedures the vehicle capacity constraint and the non-preemptive scheduling constraint are always obeyed. This guarantees the feasibility of the generated routes.

4.3 Planning phase of the half-load algorithm

In this subsection we discuss in more detail the procedure to build pseudo edges. On a tree of height 2 (Figure 2), a positive edge $e = (p(u), u)$ is split into $\lceil \frac{g(u)}{k} \rceil$ pseudo edges. Except the last, all pseudo edges have a load of k items. The last pseudo edge contains the residual $g(u)$ mod k items.

For trees with arbitrary heights, the pseudo edges and their links are built recursively in a bottom-up fashion. The pseudo edges are created from the leaves, and then spread to higher levels of the tree. For a positive vertex u, let L_+ and L_- consist of the pseudo and

negative tree edges from u to its children respectively. The pseudo edges incident to $p(u)$ and u are generated through the following steps.

First, we select a minimal size subset S of edges from L_+ whose load is just enough to service the edges in L_-. The edges in S and L_- are then merged together to form a new pseudo edge e_1. This merging is in the sense that linking the edges in S and L_- together as a group and setting $g(e_1)$ to be the total load of the involved edges. Let S' consist of e_1 and the remaining pseudo edges in L_+.

Second, a procedure called *merge* is applied on S' to produce a new set S'' of pseudo edges. The merge procedure repeatedly merges two arbitrary pseudo edges with loads $\leq \frac{k}{2}$. It is easy to see that each pseudo edge (except possibly one) in S'' should have load more than $\frac{k}{2}$. We then create a new edge e' between $p(u)$ and u for each edge $e \in S''$. The two edges are linked together through a children link of e' to remember where the load of e' is from. In the second phase, when the load of e' is requested, the children and group links can be used to locate all the edges ever participated in generating e (reachable from e by following the links). An example of building pseudo edges is shown in Figure 3.

4.4 Come-back-tree strategy for the k-delivery TSP on trees

The come-back-tree strategy is applied in the scenario when we are given two lists L_+ and L_- and a vertex u where the vehicle with load α is currently at. The negative edges of L_- will be serviced by the items from the positive pseudo edges of L_+ and load α of the vehicle. To ease the explanation, we assume that the tree is of height 2. This procedure also works for trees with arbitrary heights. The explanations for the general tree networks will be given later in this paper.

We maintain two pointers z_+ and z_- pointing to the heads of lists L_+ and L_- respectively. Recall that under the come-back rule, the vehicle may not be allowed to cross an edge if some condition occurs. The triggering condition is raised when the load of the next pseudo edge pointed to by z_+ fits the remaining capacity of the vehicle. In this case the vehicle will come back from delivering any new load and instead pick up the entire load from the pseudo edge pointed to by z_+. When the triggering condition is not raised, the vehicle begins to deliver all or part of its current load to the subtree pointed to by z_-. When the load of a pseudo edge is fully consumed, this edge is removed from L_+. Similarly when the subtree pointed to by z_- is fully served, this edge is removed from L_-. The above process continues until L_+ or L_- becomes empty. The procedure then returns the route constructed and the remaining edges in L_+ or L_-.

We explain the algorithm in [7] (which we call the full-load algorithm) as follows. For a vertex u, let $L_+ = \{c_1^+, c_2^+, \cdots, c_i^+\}$ and $L_- = \{c_{i+1}^-, \cdots, c_l^-\}$ be two lists consisting of the positive and negative children of u respectively. We also assume that in L_+ and L_- the vertices and the subtrees rooted at these vertices are sorted arbitrarily. In the full-load algorithm, the vehicle would pick up k items at a time *consecutively* from the subtrees $T_{c_1^+}, \cdots, T_{c_i^+}$, and deliver the k items consecutively to the subtrees $T_{c_{i+1}^-}, \cdots, T_{c_l^-}$. This also means that the vehicle services the subtrees in the sorted order, and when the vehicle starts to pick up (deliver), the vehicle would continue to load (consume) items if possible.

An example is given in Figure 4 to show the effectiveness of our come-back-tree strategy. In this example, $k = 8$, the input to the algorithm is a list L_+ which contains edges $e_1, e_2, \cdots, e_5$ with loads 6, 6, 6, 6, 2 respectively, and a list L_- which contains edges e'_1, e'_2, e'_3, e'_4 with loads 5, 7, 7, 7 respectively. It is not difficult to verify that only e'_3 will be traversed twice if the come-back-tree strategy is followed. Therefore $e_1, \cdots, e_5, e'_1, \cdots, e'_4$ will be traversed 1, 1, 1, 1, 1, 1, 1, 2, 1 times respectively. If we flip the types of the vertices and apply the same rule,

then only e_1 needs to be visited twice. In this case $e_1, \cdots, e_5, e'_1, \cdots, e'_4$ will be traversed 2, 1, 1, 1, 1, 1, 1, 1 times respectively. However, if we follow the full-load algorithm, then $e_1, \cdots, e_5, e'_1, \cdots, e'_4$ will be traversed 1, 2, 2, 1, 1, 1, 2, 2, 2 times respectively. It is not difficult to see that the come-back rule outperforms the full-load strategy in this example.

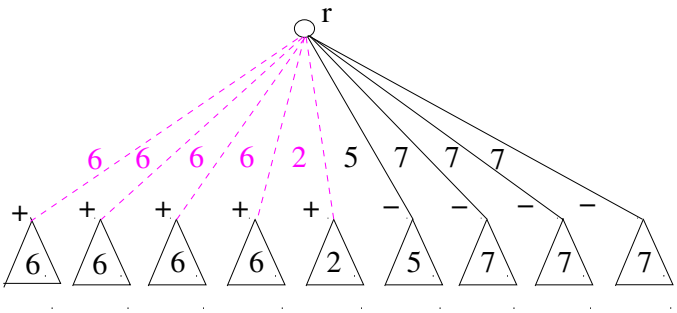


Figure 4 An example of the come-back rule with k=8.

On average, $e_1, \cdots, e_5, e'_1, \cdots, e'_4$ will be visited $\frac{3}{2}$, 1, 1, 1, 1, 1, 1, $\frac{3}{2}$, 1 times respectively in the two tours (one using G and one using $\overline{G}$) after applying the come-back rule. For trees of height 2, picking up (delivering) vertices through a pseudo edge (tree edge) can be done optimally (Figure 2). The following lemma shows that the approximation ratio of the half-load algorithm for the k-delivery TSP on trees of height 2 is $\frac{3}{2}$.

▶ **Lemma 5.** *The half-load algorithm approximates the k-delivery TSP on trees of height 2 within $\frac{3}{2}$ of its optimum.*

Proof. According to the come-back rule, each positive pseudo edge is only visited once, and the vehicle starts to deliver its load, if and only if its load plus the load of the next positive pseudo edge is more than k. Let $r_1, r_2, \cdots, r_t$ be all the routes that pass through edge $e = (p(u), u)$. We show that $t \leq 2LB_e$. It is trivially true if $t = 1$ or 2 since $LB_e \geq 1$, so we assume $t \geq 3$ in what follows. It is clear that, for $q = 1, 2, \cdots$ $(2q + 1 \leq t)$, the half-load algorithm makes the vehicle carry a total of at least $k + 1$ units from T_u in any two consecutive routes, r_{2q-1} and r_{2q}, when crossing e from $p(u)$ to u.

Case 1: t is odd. Excluding r_t, the vehicle dumped at least $(t-1)(k+1)/2 \geq (t-1)k/2+1$ items to T_u. Thus $LB_e = \lceil g(e)/k \rceil \geq \frac{t-1}{2} + 1 = \frac{t+1}{2}$, which implies that $t \leq 2LB_e - 1$.

Case 2: t is even. Excluding r_{t-1} and r_t, the vehicle dumped at least $(t-2)(k+1)/2 \geq (t-2)k/2 + 1/2$ items to T_u. Therefore $LB_e = \lceil g(e)/k \rceil \geq \frac{t-2}{2} + 1 = \frac{t}{2}$, which implies that $t \leq 2LB_e$.

Because of the symmetry, e is visited at most $3 * LB_e$ in the two solutions.

◀

We can improve the result as descirbed below. The proof is omitted.

▶ **Lemma 6.** *There is a $(\frac{3}{2} - \frac{1}{2k})$-approximation algorithm for the k-delivery TSP on trees of height 2.*

4.5 Pickup procedure for the half-load algorithm on trees of arbitrary heights

Assume L_+ and L_- contain the pseudo and tree edges respectively at the first level of the transformed tree. Our solution can be computed by invoking *come-back-tree*(α,L_+, L_-) with

$\alpha = 0$, where the first argument indicates that the vehicle is initially empty. The half-load algorithm would be complete if we have appropriate pickup and deliver procedures.

Both our pickup and deliver procedures run recursively. Given a positive pseudo edge $e = (p(u), u)$, the pickup procedure is applied when the vehicle with a load α tries to cross e to pick up $g(e)$ items from T_u. Let S be the set of pseudo edges involved in generating e, and let the pseudo edges of S induce subtree T_S (reachable from e). The vehicle should service all the delivery vertices of T_S and leave e (from u to $p(u)$) with $\alpha + g(e)$ items. This is achieved by calling the come-back-tree procedure on edges from u to its children. When this call is completed and there are still some pseudo edges unserviced, the pickup procedure will be invoked separately on these edges to collect their loads. Note that because of the non-preemptive nature, the vehicle may enter e and leave e with no items in common. An example of the pickup scheme is shown in Figure 5.

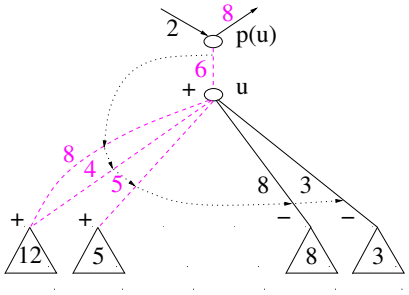


Figure 5 An example of picking up vertices from an edge $e = (p(u), u)$. $k = 8$ and the vehicle has an initial load of 2.

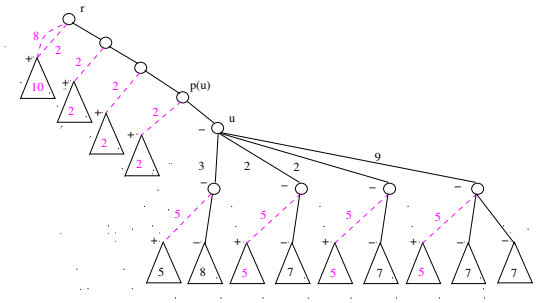


Figure 6 An example of delivering vertices to an edge $e = (p(u), u)$. $k = 8$ and r is the root of the tree.

The example in Figure 5 is a subgraph of the example in Figure 3. We want to show how to pick up 6 vertices from the first positive pseudo edge $e_1 = (p(u), u)$ in Figure 3. Before visiting e_1, the vehicle is assumed to already carry 2 vertices; since $k = 8$, the vehicle should be able to leave $p(u)$ with 8 vertices. We only consider the schedule among the edges from u to its children involved in generating e_1. Let $e_1^+, e_2^+, e_3^+, e_1^-$ and e_2^- be the 5 edges from u to its children as shown in Figure 5, respectively from left to right. In this example, the vehicle visits these edges in the order of e_1^-, e_1^+, e_1^-, e_2^+, e_2^-, e_3^+. The 8 vertices leaving $p(u)$ with the vehicle are from e_2^+ and e_3^+.

The route implied by the planning phase (a bottom-up approach) for T_S assumes zero initial vehicle load. We need to show that the tour is still feasible if the vehicle already has a load. Recall that in the come-back-tree procedure, further picking up of items through a positive pseudo edge e is allowed only if the load of e (i.e. $g(e)$) fits the remaining vehicle capacity. Therefore, when the vehicle decides to pick up $g(e)$ items, it is always guaranteed that the vehicle can service all the items on T_S and come back to $p(u)$ without violating the capacity constraint. This also shows the effectiveness of our planning phase.

4.6 Deliver procedure for the half-load algorithm on trees of arbitrary heights

The deliver procedure is applied when the vehicle with α items tries to service a negative tree edge $e = (p(u), u)$. It needs an additional parameter L'_+ which is a list of positive pseudo edges not in T_u, from which $g(e)$ items are delivered to the vertices of T_u. More specifically,

L'_+ contains the pseudo edges remaining when procedure deliver is invoked on e during a previous execution of procedure come-back-tree.

If u is a leaf node, then we simply deliver one item in the vehicle to service u. If u is not a leaf, we descend one level and calls procedure come-back-tree to service all the edges from u to its children. Different from the case for trees of height 2, some special processing is needed for trees with arbitrary heights.

First, before servicing the delivery vertices of T_u, the edges of L'_+ are attached to the end of list L_+ which consists of the positive pseudo edges from u to its children. According to the come-back-tree strategy, when the vehicle gathers the vertices from the last positive pseudo edge of the original L_+, the deliver procedure requires the information of the next positive pseudo edge of L'_+. This information will be used to decide whether the vehicle should come back and pick up some more supplies from outside T_u, to guarantee that every negative tree edge is traversed no more than twice the optimum.

Second, after the stitching of L'_+ and L_+, the merge procedure is applied on the new list before invoking procedure come-back-tree. Thus during the execution of the half-load algorithm, L_+ always contains at most one pseudo edge with load of no more than $\lceil \frac{k}{2} \rceil$. Since the algorithm runs recursively, the positive pseudo edges at higher levels of the tree are merged first. The merging, and also the way of merging in the algorithm are important for the performance guarantee. A formal proof for the correctness of the strategy is shown in Lemma 7.

▶ **Lemma 7.** *For two consecutive visits to a negative tree edge $e = (p(u), u)$ in the direction of $p(u) \to u$, the vehicle carries more than k vertices from outside T_u.*

Proof. In the algorithm, after stitching together L'_+ and L_+, the new list L_+ includes all the available positive pseudo edges (from outside T_u) which can be used to service e. For trees with arbitrary heights, the crossing of e from $p(u)$ to u might be triggered by the come-back rule when it tries to service a negative edge inside T_u. In this case the vehicle would come back and cross e from u to $p(u)$ to pick up more items from outside T_u. We claim that the pickup vertices (not in T_u) passing through e from $p(u)$ to u, are picked up consecutively from the edges of list L'_+ starting from its head, and all except possibly one of the edges in L'_+ contain more than $\lceil \frac{k}{2} \rceil$ vertices.

The first part of the claim holds, because in the deliver procedure, the edges of L'_+ is attached to the end of L_+. Therefore, after all the positive edges in the original L_+ are processed, the pickup items crossing e from $p(u)$ to u, must be picked up consecutively from the edges of list L'_+. The latter part of the claim holds, because the deliver procedure gives merging priority to higher level pseudo edges of the tree. So no matter L'_+ is obtained during the planning phase, or during an earlier call of the pickup or deliver procedure, the merging step has always been deployed on L'_+. This completes the proof.

◀

While the proof follows naturally from the algorithm, in the following we further explain why, during the delivery process, our way of merging is important for the performance guarantee. Assume before servicing a negative edge e' in T_u, the come-back rule is triggered and the vehicle crosses e from u to $p(u)$ to pick up more items, say from a pseudo edge e'' outside T_u. The vehicle collects the items from e'' and returns to T_u to service e'. Since the tree is of arbitrary height, the path P between e' and e'' may contain multiple negative edges. We need to guarantee that for all such visits (except possibly one) on an edge e, the vehicle carries more than $\frac{k}{2}$ distinct items.

Consider the tree in Figure 6. Let the four negative children of u be v_1, v_2, v_3, and v_4 respectively (from left to right). For edge $e = (p(u), u)$, $LB_e = 2$. Assume the pseudo edges in Figure 6 are not merged. Before serving the negative edge in T_{v_1}, the vehicle has a load of 8 and the first two pseudo edges in the list have loads 5 and 2 respectively. Before serving each negative edge in T_{v_2}, T_{v_3} and T_{v_4}, the vehicle has a load of 7 and the first two positive pseudo edges in the list have loads 5 and 2 respectively. It is easy to see that in this example, edge $e = (p(u), u)$ is traversed 2.5 times of LB_e if the come-back rule is deployed but the merging is not applied. If the priority of merging is not given to edges at higher levels of a tree, e.g., if each positive pseudo edge with load 5 is merged with a positive edge with load 2, then e will also be traversed 2.5 times LB_e.

▶ **Lemma 8.** *The half-load algorithm is a 2-approximation for the k-delivery TSP on trees.*

Proof. First, it is not difficult to see that the tour is feasible, since in the tour all the vertices are served and the capacity constraint is always obeyed. Given an edge e of the tree, we prove that the number of traversals the half-load algorithm makes on $e = (p(u), u)$ is no more than $2LB_e$. We have the following two cases:

Case 1: T_u is positive. Let list L_+ contain all the pseudo edges on e. Each pseudo edge e' in L_+ is traversed only once in the algorithm, thus the number of traversals the algorithm makes on e is just the number of pseudo edges in L_+. According to our merging rule in the planning phase, all pseudo edges, except possibly one, must have loads more than $\frac{k}{2}$. Assume the vehicle carries exactly $\lfloor \frac{k}{2} \rfloor + 1$ vertices each time during the first $|L_+| - 1$ visits on e. This is the worst case our algorithm could have on e. Let the last edge in L_+ be e'. When k is odd, the vehicle should carry at least $\lfloor \frac{k}{2} \rfloor + 1$ vertices during the last visit on e. In the following we show that $|L_+| \leq 2LB_e - 1$ when k is even. The case when k is odd can be argued similarly.

Subcase 1: $|L_+|$ is odd. For the first $|L_+| - 1$ visits, the lower bound of the number of traversals on e (part of LB_e) is $\frac{|L_+|-1}{2}$, if excluding $|L_+| - 1$ vertices from these visits. There must be one additional visit for these excluded vertices and the vertices in e'. So in total $LB_e \geq \frac{|L_+|-1}{2} + 1 = \frac{|L_+|+1}{2}$, which is equivalent to $|L_+| \leq 2LB_e - 1$.

Subcase 2: $|L_+|$ is even. For the first $|L_+| - 2$ visits, if excluding $|L_+| - 2$ vertices, the lower bound of the number of traversals on e (part of LB_e) is $\frac{|L_+|-2}{2}$. The vehicle must also carry more than k vertices during the last two visits, so there must also be two additional visits in LB_e. Thus in total $LB_e \geq \frac{|L_+|-2}{2} + 2 = \frac{|L_+|+2}{2}$, which is equivalent to $|L_+| \leq 2LB_e - 2$.

Case 2: T_u is negative. In this case we may have two visits on e, where the vehicle dumps less than $\frac{k}{2}$ pickup vertices on e. However, according to Lemma 7, the total vehicle load is more than k during every two consecutive traversals on e. The proof then follows much in the same way as the proof of Lemma 5. ◀

4.7 A $\frac{5}{3}$-approximation for the k-delivery TSP on trees of arbitrary heights

In this section, we prove that the smallest cost tour from three tours, obtained by applying the full-load algorithm in [7] on G (or $\overline{G}$), and the half-load algorithm on G and $\overline{G}$, is bounded by $\frac{5}{3}$ times of the optimum.

The full-load algorithm [7] has its advantages and disadvantages. The advantage is that after the vehicle gathers exactly k items, it traverses several subsequent edges optimally. The

disadvantage is that, before the vehicle is full, and after delivering these k vertices begins, some edges of the tree might have to be traversed twice of their optimum. It is not difficult to see that the half-load algorithm exchanges its advantages and disadvantages with the full-load algorithm. This explains intuitively why we balance three tours as our final solution. We formally prove the approximation ratio $\frac{5}{3}$ of our final solution in Theorem 9.

▶ **Theorem 9.** *The final solution is a $\frac{5}{3}$-approximation for the k-delivery TSP on trees of general heights.*

Proof. Given an edge $e = (p(u), u)$, let Sol_1 be the solution after applying the full-load algorithm on G (or $\overline{G}$), and Sol_2 and Sol_3 be the solutions after applying the half-load algorithm on G and $\overline{G}$ respectively. As shown in [7], e is only traversed at most $LB_e + 1$ times in Sol_1. Note here that $\overline{G}$ is obtained from G by simply relabeling the pickup vertices as delivery vertices and vice versa.

If e is positive, e is visited at most $2LB_e - 1$ times in Sol_2 according to Lemma 8. Since e is negative in $\overline{G}$, e is crossed at most $2LB_e$ times in Sol_3 (established in the proof of Lemma 5). Because of the symmetry, e is traversed at most $5LB_e$ times in the three tours for any e. ◀

5 Conclusions and future work

In this paper we propose a rule called the come-back rule for a variant of the capacitated vehicle routing problem with pickups and deliveries on trees. We illustrate this rule by an optimal algorithm for the k-delivery TSP on paths. We show this rule can be utilized to improve the approximation ratio for the k-delivery TSP on trees. Observing that the Dial-a-Ride problem [3] is also symmetric as in Lemma 3, in the future we will further investigate the Dial-a-Ride problem on trees and the k-delivery TSP on general graphs.

References

1 T. Asano, N. Katoh and K. Kawashima A New Approximation Algorithm for the Capacitated Vehicle Routing Problem on a Tree. *Journal of Combinatorial Optimization*, Vol 5, Issue 2, Page:213-231, 2001.

2 P. Chalasani, and R. Motwani, Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, Vol 28, Issue 6, Page:2133-2149, 1999.

3 M. Charikar, B. Raghavachari, The Finite Capacity Dial-a-Ride Problem. *FOCS*, Page:458-467, 1998.

4 M. Charikar, S. Khuller, B. Raghavachari, Algorithms for Capacitated Vehicle Routing. *SIAM Journal on Computing*, Vol 31, Issue 3, Page:665-682, 2002.

5 D. O. Casco, B. L. Golden, and E. A. Wasil, Vehicle Routing with Backhauls: Models, Algorithms and Case Studies. *Vehicle Routing: Methods and Studies*, Eds. Golden and Assad, North Holland, 1988.

6 M.W.P. Savelsbergh and M. Sol, The general pickup and delivery problem. Transportation Science, Vol. 29, Pages:17-29, 1995.

7 A. Lim, F. Wang, Z. Xu, The Capacitated Traveling Salesman Problem with Pickups and Deliveries on a Tree. *ISAAC* LNCS, Vol. 3827, 1061-1070, 2005.

8 F. Wang, A. Lim and Z. Xu, The One-Commodity Pickup and Delivery Traveling Salesman Problem on a Path or a Tree. *Networks* Vol. 48, Issue 1, Page:24-35, 2006.

Rerouting shortest paths in planar graphs

Paul Bonsma

Computer Science Department, RWTH Aachen University, Germany.
bonsma@informatik.rwth-aachen.de

Abstract

A rerouting sequence is a sequence of shortest st-paths such that consecutive paths differ in one vertex. We study the Shortest Path Rerouting Problem, which asks, given two shortest st-paths P and Q in a graph G, whether a rerouting sequence exists from P to Q. This problem is PSPACE-hard in general, but we show that it can be solved in polynomial time if G is planar. To this end, we introduce a dynamic programming method for reconfiguration problems.

1998 ACM Subject Classification G.2.2 (Graph algorithms)

Keywords and phrases shortest path, rerouting, reconfiguration problem, planar graph, polynomial time, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.337

1 Introduction

In this paper, we study the *Shortest Path Rerouting (SPR)* Problem, as introduced by Kamiński et al. [14]. An instance of this problem consists of a graph G with unit edge lengths, two vertices $s, t \in V(G)$, and two shortest st-paths P and Q. Shortest st-paths are *adjacent* if they differ in one vertex. The question is whether there exists a *rerouting sequence* from P to Q, which is a sequence of shortest st-paths $Q_0, \ldots, Q_k$ with $Q_0 = P$, $Q_k = Q$, such that consecutive paths are adjacent. This question may arise for instance when a commodity is routed in a network along a shortest path, and a different shortest path route is desired. However, changing the path can only be done by exchanging one node at a time, and transfer should not be interrupted [14]. A different setting where this problem may occur is if a shortest path changes randomly over time, and one wishes to know which paths are reachable from a given starting path.

On the negative side, this problem is known to be PSPACE-complete [2]. Furthermore, Kamiński et al. [14] describe instances where the minimum length of a rerouting sequence is exponential in $n = |V(G)|$. In addition they show that it is strongly NP-hard to decide whether a rerouting sequence of length at most k exists [14]. On the positive side, in [2] it is shown that SPR can be solved in polynomial time in the case where G is claw-free or chordal. In these cases, the related problem of deciding whether there is a rerouting sequence between *every* pair of shortest st-paths is also shown to be solvable in polynomial time, and in the chordal case, a shortest rerouting sequence can be found efficiently.

In this paper, we study the SPR Problem for planar graphs, and prove that this case can also be solved in polynomial time. Questions related to (rerouting) shortest paths occur often in networks, and these are often planar in practice, so this is a relevant graph class for this problem.

Similar questions about the reachability of solutions can be asked for many different combinatorial problems. This only requires defining a (symmetric) adjacency relation between feasible solutions. Ito et al. [12] called such problems *reconfiguration problems*, and initiated a systematic study of them. To be precise, for a reconfiguration problem, it is necessary

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 337–349

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

that both adjacency between solutions and the property of being a solution can be tested in polynomial time [12]. Such reconfiguration problems have been studied often in recent literature, for instance based on vertex colorings [3, 5, 6, 7], independent sets [11, 12, 15], satisfying assignments for boolean formulas [10], matchings [12], and more [9, 12, 13, 14]. Usually, the most natural adjacency relation between solutions is considered, e.g. two vertex colorings are considered adjacent in [3, 5, 6, 7] if they differ in one vertex.

One of the motivations for researching reconfiguration problems is to study the structure of the solution space of well-studied combinatorial problems, which can explain the performance of various heuristics [5, 10]. In addition, similar problems have also occurred in practical applications such as stacking problems in storage spaces [17] and train switch-yards (see [16] and references therein).

The main question that has been studied for reconfiguration problems is the complexity of deciding whether there exists a reconfiguration sequence between a given pair of solutions. For most of the aforementioned problems this problem turned out to be PSPACE-complete, although Ito et al. [12] identified a few reconfiguration problems that can be solved in polynomial time, such as matching reconfiguration. Alternatively, one may ask whether there exist reconfiguration sequences between *all* solutions [5, 6, 10], or study the question of how much the solution space needs to be increased such that a reconfiguration sequence becomes possible [4, 12], e.g. allowing to use more colors in vertex colorings used in a reconfiguration sequence [4]. In addition, one may study the problem of finding shortest reconfiguration sequences, or give upper bounds on their length [7, 10, 14].

Various advanced negative results have been proved for reconfiguration problems, such as the first two (independent) PSPACE-hardness results for such problems, on satisfiability reconfiguration [10] and sliding block puzzles [11]. To our knowledge, the other known PSPACE-hardness results on reconfiguration problems have been proved using reductions from these two results. We remark that various PSPACE-complete problems of a similar flavor have been described earlier, such as in the context of local search [18]. An essential difference is however that these are based on asymmetric adjacency relations.

In contrast, it has turned out to be rather challenging to obtain nontrivial positive results for (special cases) of reconfiguration problems. Only very few advanced polynomial time algorithms are known for reconfiguration problems, such as the algorithm by Cereceda et al. [7] on the reconfiguration of vertex colorings using three colors, and the result from Ito et al. [12] on matching reconfiguration. A reason for this lack may be that no general algorithmic techniques are known for reconfiguration problems. Introducing such techniques, and further showing that advanced positive results are possible for reconfiguration problems, are important motivations for the research presented in this paper.

We now give an outline of our results, an informal introduction to the new techniques and ideas, and sketch some obstacles for SPR in planar graphs. Detailed definitions will be given afterwards in Section 2. Firstly, in polynomial time we can delete all vertices and edges of G that do not lie on any shortest st-path, without affecting the answer. So we may assume that all vertices and edges of G lie on shortest st-paths. The problem is then straightforward in the case where the given graph G is st-planar, i.e. has a (planar) embedding where the end vertices s and t are incident with the same face, say the infinite face. The symmetric difference $\mathcal{C} := E(P)\Delta E(Q)$ of the edge sets of the two given shortest st-paths P and Q gives a set of cycles, with disjoint insides (an embedded cycle divides the plane into two regions; the finite region is called the *inside*). One can easily verify that a rerouting sequence from P to Q exists if and only if all faces inside these cycles $\mathcal{C}$ have length 4. See Figure 1(a) for an example. If G is not st-planar, there are still cases where such

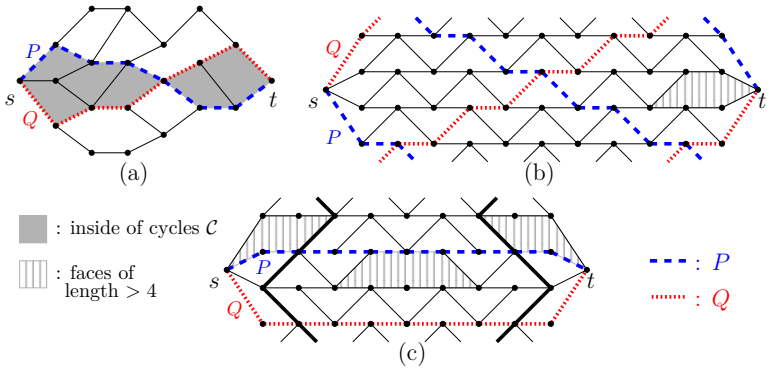


Figure 1 Three examples of planar graphs where a rerouting sequence from P to Q exists. In our figures, half edges leaving the top of the figure continue on the bottom.

a topological viewpoint works well: Figure 1(b) shows an example where every rerouting sequence $Q_0, \ldots, Q_k$ has the property that for any pair of consecutive shortest paths Q_i and Q_{i+1}, the symmetric difference of the edge sets $E(Q_i)\Delta E(Q_{i+1})$ gives a facial 4-cycle. (For readability, some edges are shown as pairs of half edges in our figures; edges leaving the top of the figure continue on the bottom.) Sloppily speaking, rerouting sequences of this type are called *topological*. Even though it is not clear in advance which region of the plane should have only faces of length 4, using some topological intuition it can be verified that also in this example a rerouting sequence from P to Q exists (the unique face of length greater than 4 is shaded). However, not all rerouting sequences in planar graphs are topological. Figure 1(c) shows an example where there exists no topological rerouting sequence from P to Q (consider the shaded faces). The method given in Section 5 will show that nevertheless, there exists a rerouting sequence from P to Q. The reason is that the two non-facial 4-cycles shown in bold can also be used for 'non-topological rerouting steps'. Cycles of this type will be called *switches*.

In Section 4, we will first give an algorithm for the problem of finding topological rerouting sequences. Next, in Section 5, switches are addressed, and our main algorithm for the SPR Problem is presented, which reduces the problem to a polynomial number of instances of the topological version of the problem. This reduction actually requires answering a slightly more general question: instead of asking for a topological rerouting sequence between two given shortest st-paths P and Q, we ask for a topological rerouting sequence from P to some shortest st-path that contains a given set of vertices. This problem is called the *Topological SPR Problem.*

The proofs in Section 5 demonstrate the following simple but powerful principle for reconfiguration problems: instead of searching for a direct reconfiguration sequence between two solutions, it is often easier to identify 'central solutions', and search for two reconfiguration sequences to a common central solution. In our case, central solutions turn out to be paths that contain many switch vertices.

To solve the Topological SPR Problem, we will not further pursue the topological ideas sketched above, but instead solve a more general problem, and develop a dynamic programming method for reconfiguration problems. This is done in Section 3, where an algorithm for the *Restricted SPR Problem* is given. In this generalization of SPR, auxiliary edges are added *(layer edges)*, to encode which rerouting steps are allowed. Our dynamic programming algorithm for Restricted SPR returns the correct answer for all instances, and does not even require the graph G to be planar, but it may require exponential time in some cases.

In Section 4, we show however that this algorithm can be used to solve the Topological SPR Problem in polynomial time. This is not the only purpose of the algorithm however; in Section 3 we also mention other nontrivial (and nonplanar) cases that it solves efficiently. The new algorithmic techniques are discussed in a broader context in Section 6, and open questions are presented. In Section 2, we first give detailed definitions. Because of space constraints, most proofs or proof details are omitted from this extended abstract; they can be found in the full version of the paper.

2 Preliminaries

For graph theoretical notions not defined here (in detail), and background on results mentioned in this section, we refer to [8]. By $N(v)$ we denote the neighborhood of a vertex v. A *walk* of length k from v_0 to v_k in a graph G is a vertex sequence $v_0, \ldots, v_k$, such that for all $i \in \{0, \ldots, k-1\}$, $v_i v_{i+1} \in E(G)$. It is a *path* if all vertices are distinct. It is a *cycle* if $k \geq 3$, $v_0 = v_k$, and $v_0, \ldots, v_{k-1}$ is a path. The corresponding subgraphs will also be called paths and cycles, respectively. A *plane graph* is a graph together with an embedding in the plane (without crossing edges). For planar graphs, an embedding can be found in polynomial time, so it suffices to prove our results for plane graphs. If a plane graph is 2-connected, then the boundary of every face is a cycle, which is called a *facial cycle*. Cycles in plane graphs correspond to simple closed curves, which divide the plane into two regions. For a cycle C in a plane graph G and vertices $s, t \in V(G)$, we say C *separates* s *from* t if s and t lie in different regions of the curve given by C (and thus $s, t \notin V(C)$). Instead of $S \cup \{x\}$ and $S \backslash \{x\}$, we write $S + x$ and $S - x$, respectively.

Throughout this paper, s and t denote two (distinct) vertices of an unweighted, undirected, simple, finite graph G, and we will consider shortest st-paths in G. Let d denote the distance from s to t in G. For $i \in \{0, \ldots, d\}$, we define $L_i \subseteq V(G)$ to be the set of vertices that lie on a shortest st-path, at distance i from s. The vertex set L_i is also called a *layer*.

Observe that a shortest st-path contains exactly one vertex of every layer, and that a shortest path is uniquely determined by its vertex set. Therefore, we will denote shortest st-paths simply by their *vertex set*. A vertex set $Q \subseteq V(G)$ such that there exists a shortest st-path P with $Q \subseteq P$ is called a *shortest st-subpath*. Note that it is not required that Q is actually a path; the vertices of Q are not necessarily from consecutive layers. Since we are only concerned with shortest paths in G between two given terminals s and t, we will call these *S-paths* for short. Shortest st-subpaths will be called *S-subpaths*. When considering reduced instances, defined by the subgraph induced by all shortest paths between two vertices x and y, these definitions refer to x and y. Given an S-path P, a rerouting step consists of replacing a vertex a by another vertex b in the same layer, such that the resulting set is again an S-path. To be precise, let $x, a, y \in P$ such that $x \in L_{i-1}$, $a \in L_i$ and $y \in L_{i+1}$. For any $b \in L_i - a$ with $\{x, y\} \subseteq N(b)$, the *rerouting step* $x, a, y \to x, b, y$ may be applied, which yields the S-path $Q = P - a + b$.

Let G be a graph and $s, t \in V(G)$. The *rerouting graph* $\mathrm{SP}(G, s, t)$ has as set of vertices all S-paths in G. Two paths are adjacent if they differ in exactly one vertex. To distinguish vertices of $\mathrm{SP}(G, s, t)$ from vertices of G, the former will be called *nodes*. Subsets of $V(\mathrm{SP}(G, s, t))$ will be called sets of S-paths or sets of nodes, depending on the context. In order to prove our results, we need to consider two additional variants of the SPR problem, which are defined by considering different adjacency relations. Call a rerouting step $x, a, y \to x, b, y$ a *restricted rerouting step* if $ab \in E(G)$. In the *restricted rerouting graph* $\mathrm{SP}^R(G, s, t)$, two S-paths P and Q are adjacent if Q can be obtained from P using a restricted

rerouting step. Edges $ab \in E(G)$ with $a, b \in L_i$ for some i are called *layer edges*.

In the case that G is a plane graph, we can define a third type of rerouting graph. A sequence of four vertices x, a, b, y is called a *switch* if x, a, y, b, x is a cycle that separates s from t, and for some i, $x \in L_i$ and $y \in L_{i+2}$. (Recall that separating s from t means that e.g. s lies 'inside' the cycle, and t 'outside'.) The vertices x and y are called its *(left and right) switch vertices*. Together, they are also called a *switch-pair*. (Note that in the switch notation x, a, b, y, the vertices are ordered non-decreasingly by their layers, but this is not the order on the cycle.) For instance, the graph G_4 shown in Figure 3 contains exactly one switch: $6, 7, 8, 9$. ($6, 8, 7, 9$ is considered to be the same switch.) A rerouting step $x, a, y \to x, b, y$ is called *topological* if x, a, b, y is not a switch. In the *topological rerouting graph* $\mathrm{SP}^T(G, s, t)$, two S-paths P and Q are adjacent if Q can be obtained from P by a topological rerouting step.

Walks in $\mathrm{SP}(G, s, t)$, $\mathrm{SP}^R(G, s, t)$ and $\mathrm{SP}^T(G, s, t)$ are called *rerouting sequences*, *restricted rerouting sequences*, and *topological rerouting sequences*, respectively. Let P be an S-path, and let Q be an S-subpath. We write $P \rightsquigarrow_G Q$ to denote that in G, there exists a rerouting sequence from P to an S-path Q' with $Q \subseteq Q'$. Similarly, the notations $P \rightsquigarrow^R_G Q$ and $P \rightsquigarrow^T_G Q$ are used for the restricted and topological case, respectively. If the graph G in question is clear, the subscript is omitted. If $P \rightsquigarrow Q$, $P \rightsquigarrow^R Q$ or $P \rightsquigarrow^T Q$, we also say that Q is *reachable* from P. We write $P \not\rightsquigarrow Q$ to denote that $P \rightsquigarrow Q$ does not hold.

The *Generalized Shortest Path Rerouting (GSPR) Problem* asks, given a graph G with $s, t \in V(G)$, an S-path P and an S-subpath Q, whether $P \rightsquigarrow_G Q$. Similarly, for the *Restricted SPR (RSPR) Problem* and *Topological SPR (TSPR) Problem*, it should be decided whether $P \rightsquigarrow^R_G Q$ and $P \rightsquigarrow^T_G Q$, respectively.

Since Q may be an *S-subpath* (in all of these problems), GSPR is a generalization of the SPR Problem (defined in Section 1). The RSPR Problem in turn generalizes the GSPR Problem: a GSPR instance G, P, Q can be transformed to an equivalent RSPR instance by adding edges between every pair of vertices in the same layer. (This may however destroy planarity.)

All algorithmic results and reductions presented in this paper for deciding $P \rightsquigarrow Q$, $P \rightsquigarrow^T Q$ or $P \rightsquigarrow^R Q$ are *constructive*, in the following sense: if $P \rightsquigarrow Q$ for an S-subpath Q, then in addition our algorithms construct an S-path Q' with $P \rightsquigarrow Q'$ and $Q \subseteq Q'$. (Analogously for the topological and restricted case.) For brevity, this is not stated in every lemma and theorem, but this fact is essential for the proofs in Section 5.

For ease of notation, we will assume throughout the paper that in the given graph G, *every vertex lies on an S-path*. Since vertices that do not lie on an S-path are irrelevant for all problems considered here, they may be deleted in advance, which can be done in polynomial time.

3 A Dynamic Programming Algorithm for RSPR

Let P be an S-path in G of length d, and let Q be an S-subpath. We want to decide whether $P \rightsquigarrow^R_G Q$. For $i = 0, \ldots, d-1$, we define the graph G_i as follows: G_i is obtained from G by first removing all vertices in layers $L_{i+1}, \ldots, L_{d-1}$, and then adding edges from t to all vertices in L_i. Using the assumption that every vertex of G lies on an S-path, in particular those in L_{d-1}, we see that $G_{d-1} = G$. By P^i and Q^i we denote $P \cap V(G_i)$ and $Q \cap V(G_i)$, which clearly are again an S-path and an S-subpath in G_i. Figure 2(a) shows an example of G_3, P^3 and Q^3, for the G, P and Q that are shown in Figure 5(a), although in Figure 5(a), we have omitted the layer edges.

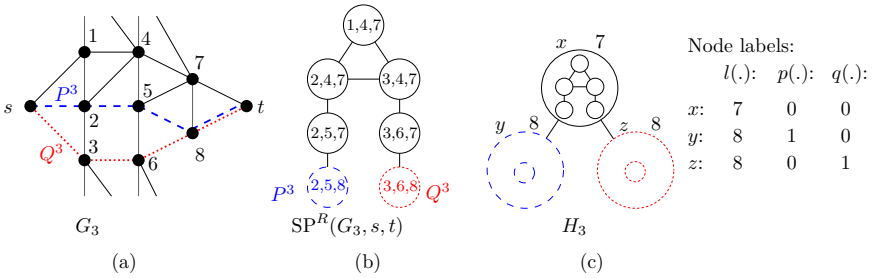


Figure 2 G_3, its restricted rerouting graph $\mathrm{SP}^R(G_3, s, t)$, and a contraction H_3 of it (with nodes x, y, z), which is the encoding of (G_3, P, Q). In the nodes of $\mathrm{SP}^R(G_3, s, t)$, the vertices of the corresponding paths are shown, except s and t. In the nodes of H_3, the corresponding contracted subgraph of $\mathrm{SP}^R(G_3, s, t)$ is drawn.

The idea is now to compute $\mathrm{SP}^R(G_{i+1}, s, t)$ from $\mathrm{SP}^R(G_i, s, t)$, for $i = 0, \ldots, d-2$. In the end, this will yield $\mathrm{SP}^R(G_{d-1}, s, t) = \mathrm{SP}^R(G, s, t)$, and we can decide whether in this graph a path from P to Q exists. The problem is of course that the graphs $\mathrm{SP}^R(G_i, s, t)$ are usually exponentially large compared to G. We solve this problem by instead considering a graph H_i that is obtained from a component of $\mathrm{SP}^R(G_i, s, t)$ by contracting connected subgraphs into single nodes, and using node labels to keep track of essential information about the corresponding path sets.

For two S-paths R and R' in G_i, we define $R \sim_i R'$ if and only if there exists a restricted rerouting sequence from R to R' that does not change the vertex in L_i (so $R \cap L_i = R' \cap L_i$). Clearly, $\sim_i$ is an equivalence relation. Furthermore, if S is an equivalence class of $\sim_i$, then S induces a connected subgraph of $\mathrm{SP}^R(G_i, s, t)$. These are exactly the subgraphs of $\mathrm{SP}^R(G_i, s, t)$ that we will contract to obtain H_i. The following definition is illustrated in Figure 2(b) and (c).

▶ Definition 1 (Encoding). Let P be an S-path in G of length d, let Q be an S-subpath in G, and let $i \in \{0, \ldots, d-1\}$. The *encoding* H_i of (G_i, P, Q) is a node-labeled graph that is obtained from $H' = \mathrm{SP}^R(G_i, s, t)$ as follows:

1. Delete every component of H' that does not contain the node P^i.
2. For every equivalence class $S \subseteq V(H')$ of $\sim_i$ that has not been deleted, contract the subgraph $H'[S]$ into a single node x, and define $S_x := S$ (this is a set of S-paths in G_i). Define $l(x)$ to be the vertex in L_i that is part of every path in S_x. Set $p(x) = 1$ if $P^i \in S_x$, and $p(x) = 0$ otherwise. Set $q(x) = 1$ if there exists an S-path $Q' \in S_x$ with $Q^i \subseteq Q'$ and $q(x) = 0$ otherwise.

H_i is now the resulting graph, together with the node labels $l(x)$, $p(x)$ and $q(x)$ for every node $x \in V(H_i)$.

Note that the encoding H_i defined this way is unique (ignoring node names). We remark that the path sets S_x for every node x are not part of the encoding and do not count towards the size; these sets are only used for the correctness proofs below. We first observe that this definition indeed allows us to decide whether $P \rightsquigarrow^R Q$.

▶ Proposition 2. Let H_{d-1} be the encoding of (G_{d-1}, P, Q). Then $P \rightsquigarrow^R Q$ if and only if H_{d-1} contains a node x with $q(x) = 1$.

Next, we study how the encoding H_{i+1} of (G_{i+1}, P, Q) is related to the encoding H_i of (G_i, P, Q). The objective is that we wish to construct H_{i+1} from H_i without considering $\mathrm{SP}^R(G_{i+1}, s, t)$. An example of this construction is shown in Figure 3. For every $v \in L_{i+1}$,

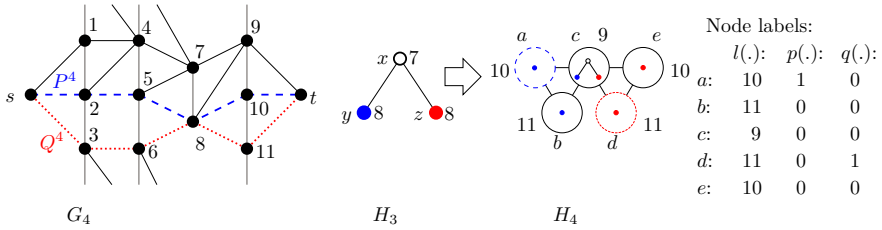


Figure 3 Constructing the encoding H_4 from H_3. In every node a of H_4, the corresponding subgraph C_a of H_3 is drawn. Numbers next to nodes a indicate their label $l(a)$.

let X_v be the set of nodes of H_i that correspond to neighbors of v. Formally, $X_v := \{x \in V(H_i) \mid l(x) \in N(v)\}$. By $H_i^v := H_i[X_v]$ we denote the subgraph of H_i induced by these nodes. This graph may have multiple components, even though H_i is connected. For $v \in L_{i+1}$ and $x \in X_v$, by $S_x \oplus v$ we denote the set obtained by adding v to every path in S_x, so $S_x \oplus v = \cup_{R \in S_x}(R + v)$. Since $l(x) \in N(v)$, this is a set of S-paths in G_{i+1}.

▶ Lemma 3. Let H_i and H_{i+1} be the encodings of (G_i, P, Q) and (G_{i+1}, P, Q), respectively. For any $v \in L_{i+1}$ and component C of H_i^v, $\cup_{x \in V(C)}(S_x \oplus v)$ is a set of S-paths in G_{i+1} that is an equivalence class of $\sim_{i+1}$. In addition, for every $a \in V(H_{i+1})$ with $l(a) = v$, there exists a component C_a of H_i^v such that $S_a = \cup_{x \in V(C_a)}(S_x \oplus v)$.

Lemma 3 shows that for every $a \in V(H_{i+1})$, there exists a corresponding component C of H_i^v, where $v = l(a)$. We denote this component by C_a.

▶ Lemma 4. Let H_{i+1} be the encoding of (G_{i+1}, P, Q). Let $a, b \in V(H_{i+1})$.

(i) $ab \in E(H_{i+1})$ if and only if $l(a)l(b) \in E(G)$ and $V(C_a) \cap V(C_b) \neq \emptyset$.
(ii) $p(a) = 1$ if and only if $l(a) \in P$ and there exists a node $x \in V(C_a)$ with $p(x) = 1$.
(iii) $q(a) = 1$ if and only if $Q \cap L_{i+1} \subseteq \{l(a)\}$ and there exists a node $x \in V(C_a)$ with $q(x) = 1$.

The previous two lemmas give all the information that is necessary to compute H_{i+1} from H_i, including the node labels l, p and q. The essential fact that will guarantee a good complexity bound is that for this computation, knowledge of the path sets S_x for $x \in V(H_i)$ is unnecessary. Together with Proposition 2, this yields a dynamic programming algorithm for deciding $P \rightsquigarrow^R Q$.

▶ Theorem 5. Let G be a graph on n vertices with two vertices $s, t \in V(G)$ at distance d. Let P be an S-path, and let Q be an S-subpath in G. In time polynomial in n and m, it can be decided whether $P \rightsquigarrow^R Q$. Here $m = \max_{i \in \{1,\ldots,d-1\}} |V(H_i)|$, where H_i is the encoding of (G_i, P, Q).

Theorem 5 shows that the RSPR problem can be solved in polynomial time if the size of the encodings H_i remains polynomially bounded. However, since the problem is PSPACE-hard [2], we should not expect this to be true for all graphs. Indeed, there exist examples where the size of the encoding grows exponentially. The example shown in Figure 4 shows that this is even true for the case of planar graphs of maximum degree 6 (or 4, when ignoring layer edges). It can be verified that for $i = 4k - 1$, the encoding H_i is a star with 2^k leaves.

However, there are many nontrivial instances for which this algorithm is polynomial. For instance, we remark (without proof) that this holds for the class of instances described by Kamiński et al. [14], where any rerouting sequence from P to Q has exponential length. (Provided that we swap the choice of s and t in their examples, or in other words, start the

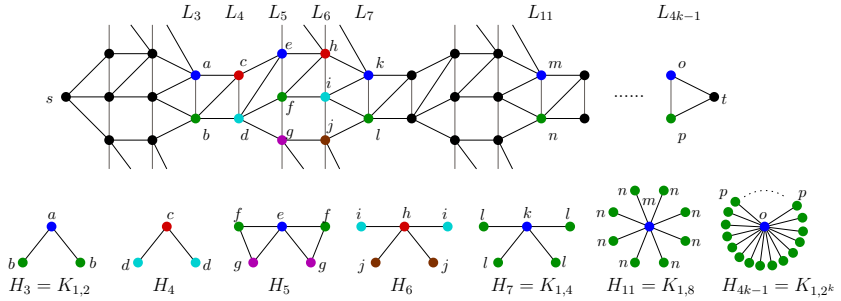


Figure 4 A plane instance of RSPR where the encoding H_i becomes exponentially large. Colors and numbers next to nodes $x \in V(H_i)$ indicate the labels $l(x)$.

computation of encodings from the side of t instead.) In addition, for the following type of low degree instances, the algorithm terminates in polynomial time. The example in Figure 4 shows that the 'degree bounds' in next result are best possible for our algorithm.

▶ Theorem 6. Let G, P, Q be an RSPR instance such that for every i and $v \in L_i$, $|N(v) \cap L_{i-1}| \leq 2$ and $|N(v) \cap L_{i+1}| \leq 2$. Then in polynomial time, it can be decided whether $P \rightsquigarrow^R Q$.

Next, we prove that H_i remains polynomially bounded for instances in a certain standard form, which is closely related to planar graphs.

4 A Polynomial Complexity Bound for TSPR

In this section, we show that if G, P, Q is a (reduced) TSPR instance, then in polynomial time it can be decided whether $P \rightsquigarrow^T Q$. To this end, we define a standard form for RSPR instances, and show for these that the algorithm from Section 3 terminates in polynomial time. Subsequently we show how TSPR instances can be transformed to equivalent RSPR instances in standard form. Figure 5(b) illustrates the following definition.

▶ Definition 7. Consider a graph G and vertices $s, t \in V(G)$ that are part of an RSPR instance. Then G is in *standard form* if the following properties hold:

(i) For every $i \in \{1, \ldots, d-1\}$, $G[L_i]$ has maximum degree 2.
(ii) For every $i \in \{1, \ldots, d-1\}$ and $v \in L_i$, $G[L_{i-1} \cap N(v)]$ is a path.
(iii) For every i and $u, v \in L_i$, if $uv \in E(G)$ then $|N(u) \cap N(v) \cap L_{i-1}| \leq 1$.

If G is in standard form, then with some effort we can show that all encodings H_i are paths, on at most $i \cdot |L_i|$ nodes. Theorem 5 then shows that our dynamic programming algorithm terminates in polynomial time. This gives:

▶ Theorem 8. Let G, P, Q be an RSPR instance in standard form. Then in polynomial time, it can be decided whether $P \rightsquigarrow^R Q$.

The objective is to apply Theorem 8 for TSPR instances G, P, Q. To this end, we will give a polynomial transformation to an equivalent instance G', P, Q of RSPR, and prove that the latter instance is in standard form. In the case of TSPR and GSPR, it will be useful to work with reduced instances. An instance G, P, Q of GSPR or TSPR is *reduced* if:

1. Every vertex and edge of G lies on an S-path,
2. G contains no cut vertices, and

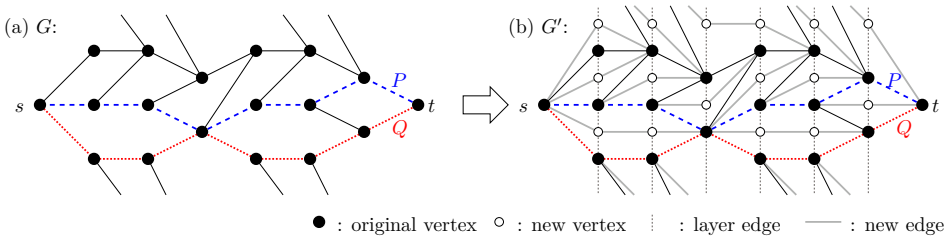


Figure 5 The transformation of a TSPR instance G, P, Q to an equivalent RSPR instance G', P, Q in standard form.

3. G contains no *neighborhood-dominated vertices*, which are vertices z for which there exists a vertex z' with $N(z) \subseteq N(z')$.

We remark that, even though the above definition is useful for both GSPR and TSPR, only in the case of GSPR we can give a polynomial time procedure that can transform every instance to a set of equivalent reduced instances. This procedure is straightforward: we may simply delete all vertices and edges not on S-paths. Next, as long as there exists a neighborhood-dominated vertex z, we may delete z, and replace occurrences of z in P and Q by the corresponding vertex z'. When a cut vertex v is present, the instance basically consists of two independent instances: one induced by all shortest sv-paths, and one induced by all shortest vt-paths.

▶ Theorem 9. Let G, P, Q be a GSPR instance. In polynomial time, a set of reduced GSPR instances can be constructed such that $P \rightsquigarrow_G Q$ if and only if every reduced instance is a YES-instance. If G is plane, all of the reduced instances are plane. The sum of the number of edges of the reduced instances is at most $|E(G)|$.

For $v \in V(G)$, by $\text{dist}_s(v)$ we denote the distance from s to v, so $v \in L_{\text{dist}_s(v)}$. We assume that G, P, Q is reduced, so every vertex and edge of G lies on an S-path. It follows that for every edge uv, it holds that $|\text{dist}_s(u) - \text{dist}_s(v)| = 1$. Furthermore, assuming G is nontrivial, G is 2-connected. So in the case where G is plane, for every face f of G and vertex v incident with f, v has exactly two incident edges uv and vw that are also incident with f. We call v a *local maximum for* f if $\text{dist}_s(v) > \text{dist}_s(u) = \text{dist}_s(w)$, and a *local minimum for* f if $\text{dist}_s(v) < \text{dist}_s(u) = \text{dist}_s(w)$.

▶ Proposition 10. Let G be a 2-connected plane graph in which every vertex and edge lies on a shortest st-path. For every face f of G, there is exactly one local maximum and one local minimum.

Now we can define the transformation from the TSPR instance G to the RSPR instance G'. This transformation is illustrated in Figure 5, and consists of the following two steps.

1. For every face f of G, we do the following. Let u and v be the local minimum and maximum of f. Since G is simple, $\text{dist}_s(u) \leq \text{dist}_s(v) - 2$. Let $\ell = \text{dist}_s(v) - \text{dist}_s(u)$. Add $\ell - 1$ *new vertices* $x_1, \ldots, x_{\ell-1}$, drawn in the face f, and ℓ edges such that $u, x_1, \ldots, x_{\ell-1}, v$ is a path of length ℓ, drawn in face f. Clearly, this preserves planarity. Call the vertices and edges introduced this way *new vertices and edges*. The vertices and edges that were already present in G are called *original vertices and edges*.
2. For every face f in the resulting graph, we do the following. Note that f still has a unique local minimum u and local maximum v. Furthermore, for every edge xy, $|\text{dist}_s(x) - \text{dist}_s(y)| = 1$. It follows that for every $i \in \{\text{dist}_s(u) + 1, \ldots, \text{dist}_s(v) - 1\}$,

there are exactly two vertices a and b incident with f in layer L_i. Between every such pair of vertices a and b, we can add an edge ab, drawn in the face f, without destroying planarity. (Hence the new edges are layer edges.)

Call the resulting plane graph G'. It can be shown that G', P, Q is an RSPR instance in standard form. Since G contains no neighborhood-dominated vertices, for any topological rerouting step $x, a, y \to x, b, y$, it holds that x, a, y, b, x is a facial cycle. So it can be replaced by two restricted rerouting steps $x, a, y \to x, z, y \to x, b, y$ for G', where z is a new vertex. Therefore, $P \rightsquigarrow^T_G Q$ implies $P \rightsquigarrow^R_{G'} Q$. For the converse, it can be shown that rerouting steps in a restricted rerouting sequence for G' can be grouped in pairs $x, a, y \to x, z, y \to x, b, y$ where only z is a new vertex. Hence x, a, y, b, x is a facial cycle of G, and $x, a, y \to x, b, y$ is a topological rerouting step. We conclude that $P \rightsquigarrow^T_G Q$ if and only if $P \rightsquigarrow^R_{G'} Q$. The above transformation is polynomial, so applying Theorem 8 gives:

▶ Theorem 11. Let G, P, Q be a reduced TSPR instance. In polynomial time, it can be decided whether $P \rightsquigarrow^T_G Q$.

5 Reducing Switches, and an Algorithm for SPR

As shown in Figure 4, the presence of switches in a plane graph G may cause our dynamic programming algorithm to take exponential time. However, in this section we show that switches also give a lot of structural information, which can be used to obtain a polynomial time algorithm. If x, a, b, y is a switch in G, then in many cases we can reduce the problem to two smaller subproblems, defined as follows: G_{sy} is the subgraph of G induced by all vertices that lie on a shortest sy-path, and G_{xt} is the subgraph of G induced by all vertices that lie on a shortest xt-path. For an S-subpath Q, we denote $Q_{xt} = Q \cap V(G_{xt})$, and $Q_{sy} = Q \cap V(G_{sy})$. We remark that the rerouting sequences that we consider in G_{sy} (G_{xt}), consist of shortest sy-paths (resp. xt-paths). We are now ready to state the key lemma for reducing the GSPR problem, when switches are present.

▶ Lemma 12. Let G, P, Q be a plane reduced GSPR instance, such that x, y is a switch pair with $\{x, y\} \subseteq P$, and Q is one of the following:

(i) an S-path that contains x and y,
(ii) $Q = \{x', y'\}$ where x', y' is a switch pair, or
(iii) $|Q| = 1$.

Then $P \rightsquigarrow_G Q$ if and only if both $P_{sy} \rightsquigarrow_{G_{sy}} Q_{sy}$ and $P_{xt} \rightsquigarrow_{G_{xt}} Q_{xt}$.

▶ Theorem 13. Let G, P, Q be a plane reduced GSPR instance, where Q is a set containing a switch pair or a single vertex of G. Then in polynomial time it can be decided whether $P \rightsquigarrow Q$.

Proof: First, compute whether $P \rightsquigarrow^T Q$, which can be done in polynomial time (Theorem 11). If yes, then clearly $P \rightsquigarrow Q$ holds as well, and an S-path Q' with $P \rightsquigarrow Q'$ and $Q \subseteq Q'$ can be computed. (Recall that, as discussed in Section 2, all of our algorithms are constructive.) If $P \not\rightsquigarrow^T Q$, then for every switch pair x, y, we compute whether $P \rightsquigarrow^T \{x, y\}$, and if so, compute the corresponding reachable S-path that contains x and y. Since the number of switch pairs in G is polynomial (linear in fact), this can again be done in polynomial time (Theorem 11). If no switch pair is reachable, then note that we may conclude that $P \not\rightsquigarrow Q$.

Now consider a switch pair x, y with $P \rightsquigarrow^T \{x, y\}$. Let P' be the S-path with $P \rightsquigarrow^T P'$ and $\{x, y\} \subseteq P'$ that has been computed. Clearly, $P \rightsquigarrow^T P'$ implies $P \rightsquigarrow P'$ and $P' \rightsquigarrow P$. Therefore, $P \rightsquigarrow Q$ if and only if $P' \rightsquigarrow Q$. By Lemma 12, $P' \rightsquigarrow Q$ if and only if both

$P'_{xt} \rightsquigarrow Q_{xt}$ and $P'_{sy} \rightsquigarrow Q_{sy}$ hold. We decide the latter two properties recursively. This way, we can decide whether $P \rightsquigarrow Q$.

It remains to consider the complexity of this algorithm. We argued that the complexity of the above procedure, not counting the recursive calls, can be bounded by a (monotone increasing) polynomial $\text{poly}(n)$, where $n = |V(G)|$. Recall that d denotes the distance between the end vertices s and t. If there are no switch pairs (which is true in particular when $d \leq 3$), then the entire procedure terminates in time $\text{poly}(n)$.

For $d \geq 3$, we prove by induction over d that the complexity of the algorithm can be bounded by $(2d-5)\cdot\text{poly}(n)$. We have just proved the induction basis ($d = 3$), so now assume $d \geq 4$. In that case, the algorithm may consider a switch pair x, y, and reduce the problem to two instances G_{sy} and G_{xt}. The distance between the end vertices of these instances is d' and $d-d'+2$, respectively, for some $3 \leq d' \leq d-1$. Using the induction assumption and the fact that both G_{sy} and G_{xt} contain at most n vertices, we can bound the total complexity by $(2d'-5)\cdot\text{poly}(n) + (2(d-d'+2)-5)\cdot\text{poly}(n) + \text{poly}(n) = (2d-5)\cdot\text{poly}(n)$. □

Finally, we are able to prove the main result of this paper.

▶ Theorem 14. Let G be a plane graph, and let P and Q be S-paths in G. In polynomial time, it can be decided whether $P \rightsquigarrow Q$.

Proof sketch: By Theorem 9, we may assume that the instance is reduced. The proof is similar to the proof of Theorem 13. The difference is that now, for every switch pair x, y, we decide whether $P \rightsquigarrow \{x, y\}$ and $Q \rightsquigarrow \{x, y\}$, which can be done in polynomial time (Theorem 13). If the answer differs for P and Q, then we may conclude $P \not\rightsquigarrow Q$. If both $P \rightsquigarrow \{x, y\}$ and $Q \rightsquigarrow \{x, y\}$, then the problem can be reduced using Lemma 12(i), and decided recursively. If $P \not\rightsquigarrow \{x, y\}$ and $Q \not\rightsquigarrow \{x, y\}$ for *every* switch pair x, y, then it suffices to decide whether $P \rightsquigarrow^T Q$, which can be done in polynomial time (Theorem 11). □

6 Discussion

In Section 3, we introduced a dynamic programming method for reconfiguration problems, that can informally be summarized as follows: identify subgraphs G_i, that are separated from the rest of the graph by small separators L_i (in our case, the distance layers). For deciding the reconfiguration problem, detailed information about all solutions is not necessary. So based on how solutions intersect with L_i, one can identify large connected subgraphs (equivalence classes) in the solution graph of G_i, which may be contracted. This method can readily be applied to all kinds of reconfiguration problems, and different sets of separators, in particular small separators given by *tree decompositions* [1]. The challenge is however proving that the resulting encodings stay polynomially bounded. In our case, exponentially large star-like structures (Figure 4) could be avoided by restricting to the topological version of the problem. There may however be different approaches, such as based on reduction rules for the encodings. Exploring this method seems useful firstly for finding other graph classes for which SPR can be solved efficiently, beyond planar graphs and low degree graphs (Theorem 6).

More generally, this dynamic programming method seems useful for answering the following interesting open question: *Are there reconfiguration problems that are PSPACE-hard for graphs of bounded treewidth? Or are there PSPACE-hard reconfiguration problems that can be solved in polynomial time for graphs of treewidth k, for every fixed k?* In contrast to the abundance of positive results on NP-complete problems for bounded treewidth [1], to our

knowledge, not a single (nontrivial) positive or negative result is known for reconfiguration problems on bounded treewidth graphs!

The results presented in this paper can easily be extended to show that if G is planar, shortest (topological) rerouting sequences have polynomial length. Furthermore, a shortest topological rerouting sequence can be found in polynomial time. Whether this also holds for finding a shortest (non-topological) rerouting sequence remains an open question. Recall that if G is not planar, this problem is strongly NP-hard [14].

In addition, the reader may verify that our dynamic programming algorithm can be used to decide efficiently whether $\mathrm{SP}^T(G, s, t)$ is connected. Whether this can also be done efficiently for $\mathrm{SP}(G, s, t)$ is another open question.

References

1 H.L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *ICALP 1988*, volume 317 of *LNCS*, pages 105–118. Springer, 1988.

2 P. Bonsma. The complexity of rerouting shortest paths. In *MFCS 2012*, volume 7464 of *LNCS*, pages 222–233. Springer, 2012.

3 P. Bonsma and L. Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoret. Comput. Sci.*, 410(50):5215–5226, 2009.

4 L. Cereceda. *Mixing graph colourings*. PhD thesis, London School of Economics and Political Science, 2007.

5 L. Cereceda, J. van den Heuvel, and M. Johnson. Connectedness of the graph of vertex-colourings. *Discrete Appl. Math.*, 308(5–6):913–919, 2008.

6 L. Cereceda, J. van den Heuvel, and M. Johnson. Mixing 3-colourings in bipartite graphs. *European J. Combin.*, 30(7):1593–1606, 2009.

7 L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *J. Graph Theory*, 67(1):69–82, 2011.

8 R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fourth edition, 2010.

9 C.E.J. Eggermont and G.J. Woeginger. Motion planning with pulley, rope, and baskets. In *STACS 2012*, volume 14 of *LIPIcs*, pages 374–383, 2012.

10 P. Gopalan, P.G. Kolaitis, E. Maneva, and C.H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM J. Comput.*, 38(6), 2009.

11 R.A. Hearn and E.D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoret. Comput. Sci.*, 343(1–2):72–96, 2005.

12 T. Ito, E.D. Demaine, N.J.A. Harvey, C.H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoret. Comput. Sci.*, 412(12–14):1054–1065, 2011.

13 T. Ito, M. Kamiński, and E.D. Demaine. Reconfiguration of list edge-colorings in a graph. In *WADS 2009*, volume 5664 of *LNCS*, pages 375–386. Springer, 2009.

14 M. Kamiński, P. Medvedev, and M. Milanič. Shortest paths between shortest paths. *Theoret. Comput. Sci.*, 412(39):5205–5210, 2011.

15 M. Kamiński, P. Medvedev, and M. Milanič. Complexity of independent set reconfigurability problems. *Theoret. Comput. Sci.*, 439:9–15, 2012.

16 F. König and M. Lübbecke. Sorting with complete networks of stacks. In *ISAAC 2008*, volume 5369 of *LNCS*, pages 895–906. Springer, 2008.

17 F. König, M. Lübbecke, R. Möhring, G. Schäfer, and I. Spenke. Solutions to real-world instances of PSPACE-complete stacking. In *ESA 2007*, volume 4698 of *LNCS*, pages 729–740. Springer, 2007.

18 C.H. Papadimitriou, A.A. Schäffer, and M. Yannakakis. On the complexity of local search. In *STOC 1990*, pages 438–445, New York, 1990. ACM.

Space Efficient Edge-Fault Tolerant Routing

Varun Rajan

Department of Computer Science and Engineering, IIT Kanpur, India.
varunrajan09@gmail.com

Abstract

Let G be an undirected weighted graph with n vertices and m edges, and $k \geq 1$ be an integer. We preprocess the graph in $\widetilde{O}(mn)$[1] time, constructing a data structure of size $\widetilde{O}(k \cdot \texttt{deg}(v) + n^{1/k})$ words per vertex $v \in V$, which is then used by our routing scheme to ensure successful routing of packets even in the presence of a single edge fault. The scheme adds only $O(k)$ words of information to the message. Moreover, the stretch of the routing scheme, i.e., the maximum ratio of the cost of the path along which the packet is routed to the cost of the actual shortest path that avoids the fault, is only $O(k^2)$.

Our results match the best known results for routing schemes that do not consider failures, with only the stretch being larger by a small constant factor of $O(k)$. Moreover, a 1963 girth conjecture of Erdős, known to hold for $k = 1, 2, 3$ and 5, implies that $\Omega(n^{1+1/k})$ space is required by any routing scheme that has a stretch less than $2k + 1$. Hence our data structures are essentially space efficient. The algorithms are extremely simple, easy to implement, and with minor modifications, can be used under a centralized setting to efficiently answer distance queries in the presence of faults.

An important component of our routing scheme that may be of independent interest is an algorithm to compute the shortest cycle passing through each edge. As an intermediate result, we show that computing this in a distributed model that stores at each vertex the shortest path tree rooted at that node requires $\Theta(mn)$ message passings in the worst case.

1998 ACM Subject Classification E.1 Data Structures

Keywords and phrases Routing, Fault tolerant algorithms, Space efficiency, Distributed data structures, Tight bounds

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.350

1 Introduction

Computer networks are increasingly becoming distributed in nature and consequently recent years have seen extensive work in the areas of distributed algorithms and computing. In the distributed model of computation, which is a more restricted version of the heavily researched parallel model, the processors or vertices, are usually bereft of access to a common shared memory, have only limited knowledge about the network topology and need to communicate with each other through message passing. Evidently this makes the problem of efficiently routing messages of core importance to distributed computing.

The process of routing a packet involves successively passing it through a series of vertices until it reaches its destination. A routing scheme is the underlying algorithm that runs as a background process on all the vertices and manages the routing process. When a vertex receives a message, the routing scheme processes the information stored in the packet header and determines what needs to be done with the message using the routing information

[1] The notation $\widetilde{O}(h(n))$ is short for $O(h(n) \log^{O(1)} n)$

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 350–361

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

available locally at the vertex. While it is often desirable to route the packet along the shortest path to its destination, its a well known fact that such an algorithm must use at least $\Omega(n)$ space per vertex.

This has lead to extensive research pertaining to compact or sub-quadratic space routing schemes. The two most important factors to consider while designing such schemes are the space requirements per node and the stretch of the routing scheme, that is, the maximum ratio between the length of the path along which the packet is routed to the length of the actual shortest path that the scheme can have for any graph.

After a series of results (refer [2, 9] for a summary of compact routing schemes), Thorup and Zwick[9] came up with some truly remarkable results. They presented near-optimal compact routing schemes that store only $\widetilde{O}(n^{1/k})$ words of information at each vertex and have a stretch of $2k-1$ when using a handshaking procedure by which the source and destination agree on an $o(\log^2 n)$ bit header that is attached to all packets, or an increased stretch of $4k-5$ when not using handshaking.

Peleg and Upfal[8] give the corresponding lower bounds by proving that any routing scheme for general graphs with a stretch $k \geq 1$ must store at least $\Omega(n^{1+1/(2k+4)})$ bits of routing information in the network. Also a 1963 girth conjecture of Erdős[6], known to hold for $k = 1, 2, 3$ and 5, implies that $\Omega(n^{1+1/k})$ space is required by any routing scheme that has a stretch less than $2k+1$. Hence the results of Thorup and Zwick are essentially optimal.

In this paper we study the problem of *fault tolerant routing* which is but a natural extension of the problem of routing and rather practical given that real world networks are quite prone to a variety of faults. Given a positive edge weighted graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, let a message traveling from a source vertex s to its destination t be currently at vertex u and let the next vertex along the shortest path to t be v. In the event of failure of the edge (u, v), a standard routing scheme would be unable to continue routing until the fault has been rectified. Our objective therefore is to design routing schemes that can ensure that the message can still be routed to t along some other path not containing the faulty component, provided such a path exists. To this effect we store some additional information at each vertex during the preprocessing stage of the routing scheme. Upon encountering a failure, this information is added to the message header to facilitate continued routing.

The first significant work that studies the problem of computing the exact paths in the presence of single edge or vertex failures in weighted directed graphs was done by Demetrescu et al. [5]. They present a data structure of $O(n^2 \log n)$ size which can be computed in $O(mn^2)$ time and reports the shortest path avoiding a faulty edge or vertex between any pair of vertices. Bernstein and Karger[1] improved the preprocessing time to $O(mn \log n)$. But neither of these algorithms is distributed in nature, and are therefore unsuitable for routing.

Courcelle and Twigg[11, 4] study the problem of exact routing avoiding a set of known forbidden vertices in weighted graphs with bounded tree width. Given a graph with tree width k and set F of forbidden vertices, they assign labels of size $\widetilde{O}(k^2)$ bits to each vertex and use this information to aid in the routing process. This space requirement would be construed as rather large as even planar graphs have tree-widths of $\widetilde{O}(n^{1/2})$ and hence would require routing tables of size $\widetilde{O}(n)$. Gavoille and Twigg[7] further improved these results to give tables of size $\widetilde{O}(k)$ for routing in planar graphs, thereby matching the corresponding best known results for shortest path routing within logarithmic factors.

Given an undirected weighted graph with edge lengths in the range $[1, W]$, Chechik et al.[3] give an $O(kn^{1+1/k} \cdot \log n \cdot \log(nW))$ space data structure that can route a message along a path that avoids a known set of faulty edges as long as there are no more than

2 faults. This path along which the message is routed has a stretch of $O(k)$. Chechik[2] further improved these results to any arbitrary number of faulty edges F unknown to s, with the corresponding stretch factors being $\widetilde{O}(|F|^3 k)$ using routing tables of size at most $O(kn^{1/k} \cdot \texttt{deg}(v) \cdot \log n \cdot \log(nW))$ at each vertex v. We note that there has been little work, exact or otherwise, to deal with routing in presence of vertex failures in general graphs.

1.1 New results and core techniques

We present simple space efficient data structures that are used by our routing scheme to route packets even in the presence of a single edge fault.

Model of computation. While the routing process utilizes only the routing information stored locally at each node and in the packet's header and hence is distributed in nature, the computation of this information itself is performed during the preprocessing stage in a sequential manner and using quadratic space.

For the routing stage, we use a standard asynchronous distributed model of computation that restricts the size of message headers to $\widetilde{O}(1)$ bits and has no access to a central shared memory.

Basic ideas and techniques. Consider a circle on a plane with a diametral chord intersecting it at points a and b and dividing it into two segments, say $\mathcal{C}_1$ and $\mathcal{C}_2$. The core idea behind our routing scheme is that the shortest path along the circle between any pair of vertices selected from the same segment will not pass through either a or b.

Let vertices u, w lie in $\mathcal{C}_1$ with u closer to a, and likewise let vertices v, x lie in $\mathcal{C}_2$ with v closer to a. Also assume that the shortest path from u to v along the circle passes through a and that (w, x) is an edge. Now if the first edge on the path from u to v were to fail and a message at u needs to be routed to v, we need only route it along the path $u - w - x - v$. This is easily achieved by storing at each vertex, say u, its routing table as well as the edge (w, x) corresponding to each edge (u, v) incident on it.

Our results. Given a network that uses the compact routing scheme of Thorup and Zwick[9] constructed with an integer k, we give data structures that store $O(k \cdot \texttt{deg}(v))$ additional information per vertex $v \in V$ and the corresponding routing algorithm that routes messages along a path of stretch $O(k^2)$ in the presence of a single edge fault.

This routing scheme adds only $O(k)$ words of data to the message header upon encountering a fault and performs only constant amount of computations per node that lies on the path to its destination (except upon encountering the fault). In particular we prove the following theorem.

▶ **Theorem 1.** *Given a positive edge-weighted undirected graph $G(V, E)$ with $|V| = n$, $|E| = m$ and a positive integer k, there is an $\widetilde{O}(mn)$ time constructible routing scheme that stores $\widetilde{O}(k \cdot \texttt{deg}(v) + n^{1/k})$ space per vertex $v \in V$ and can route messages along a path no longer than $O(k^2)$ times the shortest path in the presence of a single edge fault. The packet passed during the routing process has a header of size $O(k + \log n)$ words.*

1.2 Comparison with existing works

Merits of our results. Compared to the other algorithms that deal with compact edge fault tolerant routing, namely of Chechik et al.[3, 2], our compact routing scheme has lower space requirements and comparable stretch, and hence is more applicable in networks which only suffer from occasional failures.

Table 1 Edge fault tolerant routing schemes

	Demetrescu et al. [5]	Chechik et al. [3]	Chechik [2]	Our results
Graph type	Digraph	Undirected	Undirected	Undirected
Stretch	1	$O(k)$	$\widetilde{O}(\lvert F\rvert^3 \cdot k)$	$O(k^2)$
Distributed?	No	No	Yes	Yes
Space required	$\widetilde{O}(n^2)$ overall	$\widetilde{O}(kn^{1+1/k} \cdot \log W)$ overall	$\widetilde{O}(kn^{1/k} \log W \cdot \texttt{deg}(v))$ per vertex v	$\widetilde{O}(k \cdot \texttt{deg}(v) + n^{1/k}))$ per vertex v
Faults handled	1	2	$\lvert F\rvert$	1

[1] k is the arbitrary integer used by the underlying compact routing scheme

[2] F is the set of faulty edges

[3] W is the ratio of the weights of the heaviest edge to the lightest edge

Our routing schemes can easily be adapted to answer distance queries in presence of faults in $O(k)$ time by using the approximate distance oracles of Thorup and Zwick[10] instead of their compact routing scheme to report distances within $O(k^2)$ times the actual distances.

Demerits of our results. Approximate algorithms are generally considered to be inferior to exact ones but our algorithms do give significant improvements in terms of space requirement over the other results that handle exact failures, particularly the ones in [11, 4, 7].

For the preprocessing stages, we use sequential algorithms, not distributed ones. But as is shown in Corollary 13, even with complete access to the routing tables at each node, the number of messages required to be communicated through the network would still be rather large and undesirable.

Table 1 gives a comprehensive summary of our results and the previous works related to edge fault tolerant routing schemes.

1.3 Organization of the Paper

We describe the preliminary concepts and notations used throughout the paper in Section 2. In Section 3 we present an algorithm for computing the shortest cycle passing through each edge. We also provide a tight bound on the number of messages required to be communicated between the nodes, if the problem were to be solved in a distributed model. In Section 4 we present an $O(k^2)$-approximate routing scheme for handling single edge failures in graphs that perform routing using the compact routing scheme of Thorup and Zwick.

2 Notations and Preliminaries

Without loss of generality we make the following assumptions throughout this paper.

- The vertices in the network know each others' addresses.
- They communicate with each other using packets containing small headers used to store the routing information.
- All shortest paths are unique. We can always break ties by either ranking paths or adding small fractional weights to the edges.
- No more than a single edge failure occurs at any time.
- Message sizes and space bounds are measured in memory words where a single word is sufficient to store the edge or vertex labels and edge weights.

Also we often simply refer to storing information at an edge, when in essence this implies doing so at both of its end-vertices.

Let $G(V, E)$ be the given undirected graph with $|V| = n$, $|E| = m$ and a static weight function $\mathcal{W} : E \to \mathbb{R}^+$ defined over its edges. Let $s, t \in V$ be the source and destination vertices, respectively and let R be the underlying routing algorithm. We then define the following generic notations.

$\mathcal{T}_u$: the shortest path tree rooted at u.
$\mathcal{T}_u(v)$: the subtree of $\mathcal{T}_u$, rooted at v.
$\mathcal{P}(u, v)$: the original shortest path between u, v.
$\delta(u, v)$: the length of the path $\mathcal{P}(u, v)$.
$\mathcal{F}(u, v)$: the first edge along $\mathcal{P}(u, v)$.
$\mathcal{L}(u, v)$: the last edge along $\mathcal{P}(u, v)$.

We also apply the subscript n to $\mathcal{P}(u, v)$ and $\delta(u, v)$ to refer to the shortest path in the presence of a faulty edge, say e which may be provided as a third argument, unless it is obvious from context. The following are some terminologies and properties that are fundamental to the paper.

▶ **Definition 2.** *Consider a packet being routed along a path, say $\mathcal{P}$ and let a component, say e be faulty. Then the* **stretch** *of $\mathcal{P}$ is defined as the ratio $|\mathcal{P}|$ to $\delta_n(u, v)$. We say the path is* **exact** *if it has a stretch of 1. The stretch of the routing scheme, say k is defined as the maximum stretch that any path in any graph can have under that scheme and such a scheme is said to be a k-approximate routing scheme.*

Stretch and space requirement are indeed the most important factors to consider when designing approximate routing schemes.

▶ **Property 3** (Optimal subpath property)**.** *Any subpath of a shortest path is also a shortest path.*

▶ **Definition 4.** *The* **mid-point** *of a path, say $\mathcal{P}(u, v)$ is defined as that virtual point on $\mathcal{P}(u, v)$ that is equidistant from both u and v. This point may lie on some edge or may itself be a vertex.*

▶ **Definition 5.** *The* **mid-edge** *of a path, say $\mathcal{P}(u, v)$ is defined as*
1. *that edge on $\mathcal{P}(u, v)$ on which its mid-point lies or*
2. *either of the edges on $\mathcal{P}(u, v)$ incident on the vertex on which its mid-point lies.*

Given an edge (u, v) and the shortest cycle through it, say $\mathcal{C}$, we denote the mid-edge of the path $\mathcal{C} \setminus (u, v)$ using the notation $\mathcal{M}(u, v)$. Based on the discussion in the previous section, one may notice that, given access to an exact shortest path oracle, the $\mathcal{M}(u, v)$ values provide a compact means of storing the shortest cycle passing through each edge. An exact routing scheme can then easily use $\mathcal{M}(u, v)$ to route from u to v in the event of failure of (u, v). But the same may not be the case when using an approximate routing scheme. Hence we define separating edges, which are essentially generalizations of mid-edges.

▶ **Definition 6.** *Given a faulty component, say (u, v) and the shortest cycle through it, say $\mathcal{C}$ consider a subset of the edges along the path $\mathcal{C} \setminus e$, given in cyclic order as $(v_1, v_2), (v_3, v_4), \ldots, (v_{2l-1}, v_{2l})$. We say that these edges are* **separating edge(s)** *if the path to be taken from v_{2i} to v_{2i+1} by the underlying routing scheme does not contain (u, v), for all $i \in [0, l]$, where v_0 is u and v_{2l+1} is v.*

Obviously the set of all the edges along the path $\mathcal{C} \setminus e$ is a valid set of separating edges, but storing them would be rather costly. Consequently our objective once the cycles are computed is to find a small set of edges that are valid separating edges. The routing scheme then simply routes from edge e_i to e_{i+1} until the packet reaches v.

Information stored in the packet header. Upon encountering a faulty edge, say e we add its separating edges to the packet header in the form of a queue, say $\mathcal{Q}_{sep}$ sorted by the order that they lie on the shortest cycle of e. We assume that the standard queue operations, namely *enqueue*, *dequeue* and a subroutine, say *makeCopy* to copy one queue to another are available.

In addition to this, the header would also contain information stored by the underlying routing scheme, which would naturally contain the field `destination` to store the destination of the message. We also add a field `rejoinVertex` to store the vertex v at which we rejoin the original path.

3 Computing and Storing Shortest Cycles

In this section, we describe an algorithm for solving the Compute Shortest Cycles problem, that is, the problem of computing the shortest cycle passing through each edge in the given graph. We only require that given access to an all pairs shortest paths (APSP) oracle, any shortest path be reported optimally in time proportional to its length. To this end, we formalize the idea of using mid-edges for efficiently storing the shortest cycles through each edge and give algorithms for computing them. We start by proving the following simple lemma.

▶ **Lemma 7.** *As G is undirected, an edge (u, v) together with $\mathcal{P}_n(u, v)$ forms the shortest cycle passing through (u, v).*

Proof. Suppose that this is not the shortest cycle passing through (u, v). Then the required shortest cycle must be formed by (u, v) and some path, say $\mathcal{P}_x(u, v)$. So $\delta_x(u, v) + \mathcal{W}(u, v) < \delta_n(u, v) + \mathcal{W}(u, v)$ implying that $\delta_x(u, v) < \delta_n(u, v)$. But this is a contradiction as $\mathcal{P}_n(u, v)$ is the shortest path between u and v that does not pass through (u, v). Hence, proved. ◀

Following along similar lines of Lemma 7, it can easily be seen that the shortest path between any two vertices on a given shortest cycle through an edge is fully contained within the cycle. The following lemma utilizes this property to formalize the idea of using mid-edges for computing the shortest cycle.

▶ **Lemma 8.** *Given access to an all pairs shortest path oracle, the shortest cycle passing through each edge can be computed by storing $\mathcal{M}(u, v)$ for each $(u, v) \in E$.*

Proof. Suppose that the edge (u, v) is faulty. Let (w, x) be $\mathcal{M}(u, v)$. Consider the shortest path from u to w. As noted above, this path must either be $\mathcal{P}_n(u, w)$ or $\mathcal{C} \setminus \mathcal{P}_n(u, w)$. Now $\delta_n(u, w) \leq \frac{1}{2}\delta_n(u, v)$ since (w, x) is $\mathcal{M}(u, v)$ and lies on $\mathcal{P}_n(u, v)$. Then $|\mathcal{C}| - |\mathcal{P}_n(u, w)| \geq |\mathcal{C}| - \frac{1}{2}\delta_n(u, v) \geq \mathcal{W}(u, v) + \frac{1}{2}\delta_n(u, v) > \frac{1}{2}\delta_n(u, v)$. Hence $\mathcal{P}_n(u, w)$ is the same as $\mathcal{P}(u, w)$. Similarly we can prove that $\mathcal{P}_n(v, x)$ is the same as $\mathcal{P}(v, x)$, and we can report the shortest cycle through (u, v) as $\mathcal{P}(u, w) \cup (w, x) \cup \mathcal{P}(v, x)$. Hence, proved. ◀

Data structure. As characterized by Lemma 8, we store $\mathcal{M}(u, v)$ for all $(u, v) \in E$. We also store the corresponding $\delta_n(u, v)$ values for obtaining the length of the shortest cycles. Only $O(1)$ words per edge are required for storing these values.

Algorithm. Algorithm SHORTESTCYCLES (See Appendix A.1 for pseudo-code) first computes the APSP oracle using Dijkstra's algorithm and then invokes `computeCycles()` on each edge. For the current edge, say (u, v) it checks for each vertex $w \in V$ if $\mathcal{F}(u, w) \neq \mathcal{F}(v, w)$ and $\mathcal{L}(u, w) \neq \mathcal{L}(v, w)$. If either of these checks fail, the optimal subpath property implies that (u, v) does not form a cycle with the paths $\mathcal{P}(u, w)$, $\mathcal{P}(v, w)$. Upon finding a new cycle, say $\mathcal{C}_1$ the algorithm checks if this cycle is shorter than the currently recorded shortest cycle. As the mid-edge must lie on the longer path amongst $\mathcal{P}(u, w)$ and $\mathcal{P}(v, w)$, The algorithm further checks if the mid-point of the path $\mathcal{C}_1 \setminus (u, v)$ lies on either of the edges adjacent to w, and updates $\delta_n(u, v)$ and $\mathcal{M}(u, v)$ if required.

Simply looping over all the vertices in this manner only considers edges in $\mathcal{T}_u$ and $\mathcal{T}_v$ to be mid-edges. As is shown in Lemma 11, the mid-edge of $\mathcal{P}(u, v)$ need not be present in either of these shortest path trees, and therefore `computeCycles()` cannot guarantee that the shortest cycle is correctly determined. Hence, whenever a new cycle is computed by `computeCycles()`, it invokes the `updateCycles()` procedure which further checks if edge (u, v) can be the mid-edge of the two edges along the cycle that are adjacent to w. Lemma 12 shows that this addition is sufficient to ensure that the correct shortest cycles are computed for each edge when the algorithm terminates. The complete pseudo-code for these procedures is given in Appendix A.1.

▶ Remark. It is worth noting that we can easily loop over the edges in a distributed manner. But before running the `updateCycles()` routine, we will then have to send messages to inform vertex w of the existence of edge (u, v). This would need to be done every time a new cycle is computed, leading to a rather large message complexity of $O(mn)$.

Correctness. We are required to prove that the correct values of $\mathcal{M}(u, v)$ are computed for each edge $(u, v) \in E$ when the SHORTESTCYCLES algorithm terminates. To this effect, we need to ensure that the algorithm loops over every cycle that can be the shortest cycle of some edge before it terminates. As the number of cycles that an edge is a part of can be quite large, often going into the exponentials of n, we need to characterize those cycles which can be a shortest cycle to make the search space tractable.

For simplicity, we say that a cycle is a *candidate shortest cycle* of the given edge if it can be the shortest cycle passing through it, and that such a cycle is *reported* if the algorithm checks to see whether it is shorter than the currently recorded shortest cycle. Naturally when a candidate shortest cycle gets reported, any longer candidate shortest cycles that are not yet reported, cease to be candidates. To reiterate, our objective is to report every candidate shortest cycle. The following observation helps substantially reduce the search space.

▶ **Observation 9.** *From the definition of mid-point, it follows that every path contains at least one mid-edge (two, if the mid-point lies on a vertex). By Lemma 8, we can then express the shortest cycle through each edge $(u, v) \in E$ as an union of two shortest paths and their connecting edges, as $(v, u) \cup \mathcal{P}(u, w) \cup (w, x) \cup \mathcal{P}(x, v)$, where (w, x) is $\mathcal{M}(u, v)$.*

This observation reduces the set of candidate shortest cycles of $(u, v) \in E$ to those cycles $\mathcal{C}$, with $(w, x) \in E$ being the mid-edge of $\mathcal{C} \setminus (u, v)$, that are formed by the concatenation, at both ends, of edge-disjoint shortest paths $\mathcal{P}(u, w)$ and $\mathcal{P}(v, x)$. Thus we need only iterate over every pair of edges to determine the shortest cycles, which is still rather costly. Here we make another crucial observation that given such a cycle $\mathcal{C}$ if the edge (w, x) were to belong to the path $\mathcal{P}(u, x)$ (or to $\mathcal{P}(v, w)$), then we can get sufficient information to determine $\mathcal{C}$ by going over the vertex x (or w) instead of the edge (w, x). This intuitively leads us to consider cycles formed by every edge-vertex pair and characterize those candidate shortest cycles of (u, v) that cannot be determined by this approach.

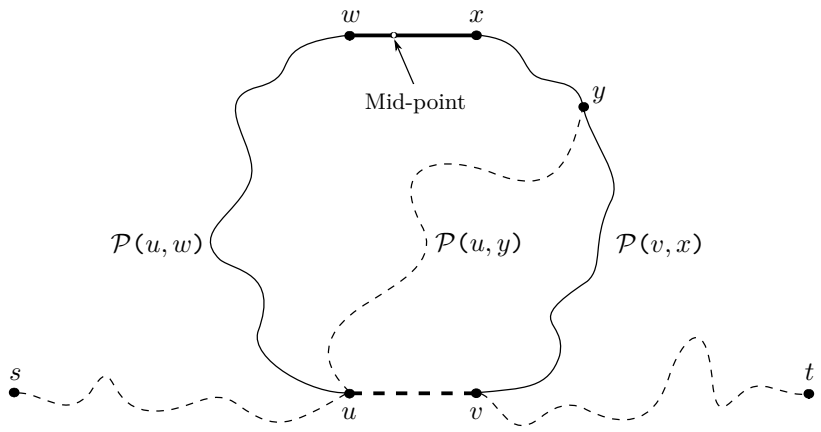


Figure 1 Cycle passing through an edge

For the remainder of this section, we will use the premise that we are given an edge $(u,v) \in E$ and a cycle, say $\mathcal{C}$ passing through it ($\mathcal{C}$ need not be the shortest cycle through it). Without loss of generality, assume that the mid-point of the path $\mathcal{C} \setminus (u,v)$ lies on an edge $(w,x) \in E$. By the definition of mid-edges, it then follows that $\mathcal{P}(u,w)$, (w,x), $\mathcal{P}(x,v)$ and (v,u) together constitute $\mathcal{C}$ (refer fig. 1). Thus, by Observation 9, $\mathcal{C}$ is indeed a representative of all possible candidate shortest cycles. Note that, in the `computeCycles()` procedure, only cycles that are expressed in this manner are checked to see if they are shorter than the currently recorded shortest cycle. Lemma 10 now characterizes the candidate shortest cycles that will be reported by `computeCycles()`.

▶ **Lemma 10.** *If at least one of the paths $\mathcal{P}(u,x)$ and $\mathcal{P}(v,w)$ passes through (w,x), then cycle $\mathcal{C}$ will get reported by* `computeCycles()`.

Proof. Assume without loss of generality that path $\mathcal{P}(u,x)$ passes through (w,x). Then by the optimal subpath property, $\mathcal{P}(u,x)$ is the same as $\mathcal{P}(u,w) \cup (w,x)$ and hence does not intersect $\mathcal{P}(v,x)$, implying that cycle $\mathcal{C}$ will be reported by `computeCycles(u,v)` when it loops over the vertex x. Hence, proved. ◀

So cycle $\mathcal{C}$ is not reported only if neither $\mathcal{P}(u,x)$ nor $\mathcal{P}(v,w)$ contain (w,x). Lemma 11 further characterizes these candidate shortest cycles of (u,v) that are not reported.

▶ **Lemma 11.** *If cycle $\mathcal{C}$ is not reported by* `computeCycles(u,v)`*, then the following statements are equivalent.*

1. *$\mathcal{C}$ is a candidate shortest cycle of (u,v).*
2. *The paths $\mathcal{P}(u,x)$ and $\mathcal{P}(v,w)$ both pass through (u,v).*
3. *w and x belong to different subtrees in both $\mathcal{T}_u$ and $\mathcal{T}_v$.*

Proof. We prove the statements in a cyclic order.

1 ⇒ 2: We prove the contrapositive of this statement which is stated below.

If $\mathcal{C}$ is not reported by `computeCycles()` *and at least one of the paths $\mathcal{P}(u,x)$ and $\mathcal{P}(v,w)$ does not pass through (u,v), then $\mathcal{C}$ is not a candidate shortest cycle of (u,v).*

Suppose $(u,v) \notin \mathcal{P}(u,x)$. As $\mathcal{C}$ is not reported, $(w,x) \notin \mathcal{P}(u,x)$. Then $\mathcal{P}(u,x)$ intersects path $\mathcal{P}(v,x)$ at some vertex, say y that lies on the path from v to x, both inclusive (refer fig. 1). So $\delta(u,y) + \delta(v,y) \leq \delta(u,x) + \delta(v,x) < \delta(u,w) + \mathcal{W}(w,x) + \delta(v,x)$. This shows that $\mathcal{P}(u,y)$, $\mathcal{P}(v,y)$ and (u,v) together form a cycle shorter than $\mathcal{C}$, implying that $\mathcal{C}$ is not a candidate shortest cycle. Similarly we can prove for the case that $(u,v) \notin \mathcal{P}(v,w)$.

2 ⇒ 3: As $\mathcal{P}(u,w)$, (w,x), $\mathcal{P}(x,v)$ and (v,u) together form $\mathcal{C}$ we have $\mathcal{F}(u,w) \neq (u,v)$. Also from statement 2, $\mathcal{F}(u,x) = (u,v)$. So $\mathcal{F}(u,w) \neq \mathcal{F}(u,x)$ implying that w and x belong to different subtrees in $\mathcal{T}_u$. We can similarly prove for $\mathcal{T}_v$.

3 ⇒ 1: From statement 3 we get that edge (w,x) is not present in either $\mathcal{T}_u$ or $\mathcal{T}_v$. This means that `computeCycles(u,v)` never considers the edge (w,x) or consequently, the cycle $\mathcal{C}$. But $\mathcal{C}$ can be shorter than the shortest cycle of (u,v) recorded by it and hence needs to be reported. This concludes the proof as any cycle that needs to be reported is a candidate shortest cycle by definition.

Hence, proved. ◀

Lemmas 10 and 11 thus analyze all the candidate shortest cycles of (u,v) and it turns out that a cycle would still need to be reported for (u,v) if and only if both the path $\mathcal{P}(u,x)$ and $\mathcal{P}(v,w)$ pass through it. All that remains now is to prove that these cycles will get reported by the `updateCycles()` procedure before the algorithm terminates.

▶ **Lemma 12.** *Every cycle which is a candidate shortest cycle is reported, and $\delta_n(u,v)$ and $\mathcal{M}(u,v)$ computed for each edge $(u,v) \in E$ by the algorithm.*

Proof. Suppose that a candidate shortest cycle of (u,v) is not reported. From statement 3 of Lemma 11, we know that w and x belong to different subtrees of $\mathcal{T}_u$, implying that $\mathcal{F}(w,u) \neq \mathcal{F}(x,u)$ and $\mathcal{L}(w,u) \neq \mathcal{L}(x,u)$. But this is the same condition check that is performed when the algorithm loops over the vertex u while computing the shortest cycle through (w,x). Hence the check will pass here, `updateCycles()` procedure will be called with the arguments (u,w,x), and thus the cycle will get reported for (u,v).

The routines used for computing $\delta_n(u,v)$ and $\mathcal{M}(u,v)$ are standard, follow along the same lines as a sequential algorithm for finding the minimum element in an array, and their correctness is easy to see. Hence, proved. ◀

Lemma 11 has an important implication on the efficiency of algorithms that compute shortest cycles in a distributed model that stores at each node the shortest path tree rooted at that node. As each node, say u is unaware of $O(m)$ edges that are not present in $\mathcal{T}_u$ and since the shortest cycle passing through it could be formed together with any of these edges, each of these edges need to inform u of their existence. In the worst case, this would result in $\Omega(mn)$ messages being generated. It is for this reason that we do not perform the preprocessing step using a distributed algorithm. Moreover our algorithm will require $O(mn)$ message passings as described earlier and hence we have the following corollary.

▶ **Corollary 13** (of Lemmas 11, 12)**.** *The* Compute Shortest Cycles *problem requires $\Theta(mn)$ messages passings to compute the shortest cycle through each edge in a distributed model of computing that stores at each node the shortest path tree rooted at that node.*

Time and Space complexities. Computing the APSP oracle requires $O(mn+n^2 \log n)$ time, while the other computations take only $O(mn)$ overall time. Hence the algorithm has an overall time complexity of $O(mn + n^2 \log n)$. We require $O(n^2)$ space for storing the APSP oracle and $O(m)$ space for the output.

Theorem 14 follows directly from the results given in this section.

▶ **Theorem 14.** *Given a graph $G = (V,E)$ with $|V| = n$, $|E| = m$, there is an algorithm that can compute $\delta_n(u,v)$ and $\mathcal{M}(u,v)$ for each $(u,v) \in E$ in $\widetilde{O}(mn)$ steps using $O(n^2)$ space.*

4 An $O(k^2)$-Approximate Edge Fault Tolerant Routing Scheme

In this section we first use the data structure computed in the previous section to develop an algorithm to compute the *separating edges* corresponding to each edge. We then give a simple but space efficient routing scheme that uses this data structure to successfully route packets even in the presence of a single edge fault. While normal routing will be performed using the compact routing schemes of Thorup and Zwick[9], henceforth referred to as the R_{tz} scheme, we will continue to use the APSP oracles for preprocessing. In particular we prove Theorem 1 given in the Introduction section.

Given an integer k, the R_{tz} routing scheme computes tree covers of the given graph, which are a family of induced trees such that, between any pair of vertices in the graph, a path no longer than $4k-5$ times the shortest path exists in one of these trees. It then stores the tree cover in a distributed manner, using only $\widetilde{O}(n^{1/k})$ space per vertex and uses $o(\log n)$ size headers for routing. We particularly emphasize a feature of this scheme, namely that during the routing process, the message passes through an edge no more than once.

Recall that separating edges of an edge, say (u,v) are edges lying on $\mathcal{P}_n(u,v)$ and given in cyclic order as $e_1, e_2, \ldots, e_l$, such that the path that R_{tz} takes from e_i to e_{i+1} will not contain (u,v), for all $i \in [0, l+1]$, where $e_0 = e_{l+1} = (v,u)$. We need to compute small sets of separating edges for each edge. The following lemma helps us with this.

▶ **Lemma 15.** *Let $\mathcal{C}_e$ be the shortest cycle through an edge $e \in E$ with $|\mathcal{C}_e| = x$ units. Let $u, v \in \mathcal{C}_e$ be vertices satisfying the conditions $e \notin \mathcal{P}(u,v)$ and $4(k-1)\cdot\delta(u,v) < x$. Then the path from u to v reported by the R_{tz} scheme does not contain e either.*

Proof. Assume on the contrary that the path reported by the R_{tz} scheme from u to v contains e. As $\mathcal{P}(u,v)$ does not contain e, we get by the optimal subpath property that this path and the reported path together contain a cycle no longer than $\delta(u,v)+(4k-5)\delta(u,v) = 4(k-1)\cdot\delta(u,v)$ that passes through e. But this implies that a cycle shorter than $\mathcal{C}_e$ exists as $4(k-1)\cdot\delta(u,v) < x$, giving us a contradiction. Hence, proved. ◀

This lemma essentially implies that we need to divide each shortest cycle into $4(k-1)$ subpaths separated by single edges that are stored as the separating edges.

Data Structure. We compute and store the separating edges, in cyclic order, as a queue $\mathcal{Q}(u,v)$ for each $(u,v) \in E$. This requires $O(k \cdot \texttt{deg}(v))$ space per vertex v, as shown in Lemma 16. The information required by the R_{tz} scheme is also stored, leading to an overall space requirement of $\widetilde{O}(k \cdot \texttt{deg}(v) + n^{1/k})$ per vertex v.

Preprocessing algorithm. Given a graph $G = (V, E)$, algorithm SEPARATINGEDGES (See Appendix A.2 for pseudo-code) first computes the APSP oracle using Dijkstra's algorithm as before and then runs `computeCycles()` on the graph to compute $\mathcal{M}(u,v)$ and $\delta_n(u,v)$ for each edge $(u,v) \in E$. It then invokes the `compute-kSep()` procedure for each edge to compute separating edges along the shortest cycle passing through it.

Given an edge (u,v) and the corresponding $\delta_n(u,v)$, `compute-kSep()` successively computes the longest section shorter than $\frac{\delta_n(u,v)}{4(k-1)}$ units and adds the next edge to $\mathcal{Q}(u,v)$ as a separating edge. The complete pseudo-code for these procedures is given in Appendix A.2.

Correctness. The correctness of the shortest cycle computing routines have already been proved in the previous section. Procedure `compute-kSep()` trivially ensures that the subpath between two separating edges of (u,v), with indices i and $i+1$ in $\mathcal{Q}$, are shorter than $\frac{\delta_n(u,v)}{4(k-1)}$ and thus satisfies the requirements implied by Lemma 15. Hence the edges stored in $\mathcal{Q}$ are valid separating edges. Next we bound the size of the queue $\mathcal{Q}$.

▶ **Lemma 16.** *$\mathcal{Q}$ contains only $O(k)$ elements per edge $e \in E$.*

Proof. Given an edge $(u, v) \in E$, each section computed by `compute-kSep()` together with its corresponding separating edge is longer than $\frac{\delta_n(u,v)}{4(k-1)}$. As the path is only $\delta_n(u, v)$ long, there can be no more than $4(k-1)$ separating edges. Hence, proved. ◀

Time and Space complexities. It takes $\widetilde{O}(mn)$ time for computing the APSP oracle and the shortest cycles. For each edge (u, v), `compute-kSep()` traverses the path $\mathcal{P}(u, v)$ performing $O(1)$ computations per vertex along the path. As there can be no more than $n-2$ other vertices along the path, `compute-kSep()` performs $O(mn)$ computations only. Hence the algorithm has an overall time complexity of $\widetilde{O}(mn)$.

The algorithm requires $O(n^2)$ space for storing the APSP oracle, and δ_c and $\mathcal{M}$ require $O(1)$ space per edge. The queues require $O(k \cdot \texttt{deg}(v))$ space per vertex v. Hence the overall space complexity of SEPARATINGEDGES is $O(n^2)$. We have the following theorem.

▶ **Theorem 17.** *Given a graph $G(V, E)$ with $|V| = n$, $|E| = m$, there is an algorithm that can compute the queue of separating edges $\mathcal{Q}(u,v)$ for each edge $(u, v) \in E$ in $\widetilde{O}(mn)$ steps using $O(n^2)$ space. The output, $\mathcal{Q}$ requires $O(k \cdot \texttt{deg}(v))$ space per vertex v, where k is the integer used for constructing the underlying R_{tz} routing scheme.*

4.1 The Routing Scheme

The routing is performed normally using the R_{tz} routing scheme until the faulty edge, say (u, v) is encountered. $\mathcal{Q}(u,v)$ is then copied to $\mathcal{Q}_{sep}$ in the packet header. The routing is then performed by dequeueing an edge, say (w, x) from $\mathcal{Q}_{sep}$ and routing normally to the dequeued edge. This is repeated until the vertex v is encountered. From here normal routing resumes. The complete routing scheme, described in a step-by-step format follows.

1. Route normally until reaching the `destination` or a faulty component, say e.
2. Copy $\mathcal{Q}(e)$ to $\mathcal{Q}_{sep}$ and set v as `rejoinVertex` in the packet header.
3. Dequeue from $\mathcal{Q}_{sep}$ and route to the dequeued edge.
4. Repeat *Step 3* until $\mathcal{Q}_{sep}$ is empty.
5. Route to `rejoinVertex`.
6. Continue normal routing to `destination`.

The correctness of the routing scheme follows directly from the definition of separating edges. Lemma 18 bounds the stretch of this routing scheme.

▶ **Lemma 18.** *The path taken by the routing scheme has a worst-case stretch of $O(k^2)$, where k is the integer used to construct the R_{tz} routing scheme.*

Proof. Consider a packet going from vertex s to t that encounters a faulty edge, say e while traversing along a path no longer than $(4k-5) \cdot \delta(s, t)$. This path together with either the original or the new shortest paths contains a cycle no longer than $(4k-5) \cdot \delta(s, t) + \delta_n(s, t) \leq 4(k-1) \cdot \delta_n(s, t)$. The subpaths between successive separating edges of e along this cycle are further approximated by a factor no more than $4k-5$ as we again use the compact routing scheme. Then the overall distance over which the packet is routed, say X is given as follows.

$$\begin{aligned} X &< (4k-5) \cdot \delta(s,t) + (\text{No. of sections}) * [4(k-1) \cdot \delta(s,t)] \\ &\leq (4k-5) \cdot \delta(s,t) + 4(k-1) * [4(k-1) \cdot \delta(s,t)] \\ &< 16k^2 \delta_n(s,t) \end{aligned}$$

implying that the stretch of the routing scheme is no more than $O(k^2)$. Hence, proved. ◀

Theorem 17 and Lemma 18 together conclude the proof of Theorem 1.

5 Conclusions and Open Problems

The problem of routing in presence of failures has received considerable attention in recent years. We employ an innovative and simple approach to the problem, and present new results that are space-efficient, essentially optimal and match the best known results for compact routing that do not handle failures within a small $O(k)$ factor of stretch. There are several research problems that merit further study.

1. The stretch factor of $O(k^2)$ for our compact routing scheme, may be construed as rather large and undesirable in practice. By properly integrating the routing scheme of Thorup and Zwick rather than using it as a mere black-box, it may very well be possible to significantly reduce the stretch.
2. While the results of Chechik et al.[3, 2] as well as ours give compact space routing schemes for the case of edge failures, such results are not known for the case of vertex faults. Designing compact vertex fault tolerant routing schemes would be quite significant.
3. It would be interesting to see if our techniques can be extended to the case of multiple faults. This work would be in the same vein as that of the multiple fault tolerant scheme of Chechik[2].
4. Extending the algorithms to a streaming/dynamic settings would greatly increase their practicality.

Acknowledgements

We thank the anonymous reviewers for their valuable and detailed comments on improving the presentation.

References

1 Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *STOC*, pages 101–110, 2009.
2 Shiri Chechik. Fault-Tolerant Compact Routing Schemes for General Graphs. In *ICALP (2)*, pages 101–112, 2011.
3 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. *f*-Sensitivity Distance Oracles and Routing Schemes. In *ESA (1)*, pages 84–96, 2010.
4 Bruno Courcelle and Andrew Twigg. Compact Forbidden-set Routing. In *STACS*, volume 4393 of *LNCS*, pages 37–48, 2007.
5 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.
6 P. Erdős. Extremal problems in graph theory. In *Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)*, pages 29–36, 1964.
7 Cyril Gavoille and Andrew Twigg. Compact Forbidden-set Routing on Planar Graphs. Unpublished as yet.
8 David Peleg and Eli Upfal. A Trade-Off between Space and Efficiency for Routing Tables. *J. ACM*, 36(3):510–530, 1989.
9 Mikkel Thorup and Uri Zwick. Compact Routing Schemes. In *Proc. 13th ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2001.
10 Mikkel Thorup and Uri Zwick. Aproximate Distance Oracles. *J. ACM*, 52:1–24, 2005.
11 Andrew Twigg. *Compact forbidden-set routing.* Tech. Rep. UCAM-CL-TR-678, University of Cambridge, December 2006.

Approximate Determinization of Quantitative Automata*

Udi Boker[1,2] and Thomas A. Henzinger[2]

1 **Hebrew University of Jerusalem**
2 **IST Austria**

Abstract

Quantitative automata are nondeterministic finite automata with edge weights. They value a run by some function from the sequence of visited weights to the reals, and value a word by its minimal/maximal run. They generalize boolean automata, and have gained much attention in recent years. Unfortunately, important automaton classes, such as sum, discounted-sum, and limit-average automata, cannot be determinized. Yet, the quantitative setting provides the potential of approximate determinization. We define approximate determinization with respect to a distance function, and investigate this potential.

We show that *sum* automata cannot be determinized approximately with respect to any distance function. However, restricting to nonnegative weights allows for approximate determinization with respect to some distance functions.

Discounted-sum automata allow for approximate determinization, as the influence of a word's suffix is decaying. However, the naive approach, of unfolding the automaton computations up to a sufficient level, is shown to be doubly exponential in the discount factor. We provide an alternative construction that is singly exponential in the discount factor, in the precision, and in the number of states. We prove matching lower bounds, showing exponential dependency on each of these three parameters.

Average and *limit-average* automata are shown to prohibit approximate determinization with respect to any distance function, and this is the case even for two weights, 0 and 1.

1998 ACM Subject Classification F.1.1 Automata; D.2.4 Formal methods

Keywords and phrases Quantitative; Automata; Determinization; Approximation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.362

1 Introduction

Quantitative automata have gained much attention in recent years. In the scope of formal verification, they allow a generalization of the traditionally boolean setting into a quantitative one (e.g. [5, 10]). They are nondeterministic finite-state automata with edge weights, valuing a run by a function from the sequence of visited weights into $\mathbb{R}$. Unlike weighted automata that are based on a semi-ring [12] or lattice [14], quantitative automata do not limit the value function to certain algebraic properties. A quantitative automaton $\mathcal{A}$ assigns to each word w a real value $\mathcal{A}(w)$, which is the infimum or the supremum of all runs of the automaton on w.

Automata determinization is fundamental in formal methods, such as synthesis, which is based on game solving, and verification, which is based on the comparison of automata. Game solving requires automata that are essentially deterministic [13], and checking for automata

* This work was supported in part by the Austrian Science Fund NFN RiSE (Rigorous Systems Engineering) and by the ERC Advanced Grant QUAREM (Quantitative Reactive Modeling).

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 362–373

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

equivalence and inclusion has good algorithms for deterministic automata, whereas it is expensive for boolean nondeterministic automata and even undecidable for some quantitative nondeterministic automata (such as limit-average automata [11]). Unfortunately, important quantitative automaton classes, such as sum, average, discounted-sum, LimSup, and limit-average automata, cannot be determinized [8].

The quantitative setting is directly related to approximation. Having real values for words, naturally suggests to approximate one automaton by another, e.g. a nondeterministic automaton by a deterministic one, requiring that the latter's values on all words are close enough to the former's values. With such approximately determinized automata, one may approximately compare between the original nondeterministic automata, as well as approximately solve games.

How should one define "close enough"? A general choice is some distance function on $\mathbb{R}$. We therefore define that an automaton $\mathcal{A}$ can be *determinized approximately* with respect to a distance function d if for every real precision $\varepsilon > 0$, there is a deterministic automaton $\mathcal{D}$ such that for every word w, $d(\mathcal{A}(w), \mathcal{D}(w)) \leq \varepsilon$.

One may check whether some automaton type can be determinized approximately with respect to a specific distance function, for example with respect to the absolute difference function ($|\mathcal{A}(w) - \mathcal{D}(w)|$). In addition, seeking a thorough picture of approximate determinization, one may try to show that a certain automaton type cannot be determinized approximately with respect to any distance function in a certain class. A central observation is that quantitative automata relate to the natural order on $\mathbb{R}$, as they use non-determinism for choosing the infimum or supremum of the run values. For that reason, an interesting and general class of distance functions (called *ordered* distance functions) contains those that respect the order on $\mathbb{R}$, meaning that for every $x \leq y \leq z \in \mathbb{R}$, it holds that $d(x, y) \leq d(x, z)$ and $d(y, z) \leq d(x, z)$.

The importance of approximate determinization is well known (e.g. [4, 7]), yet central questions are still open. In [4], they consider approximate determinization of sum automata with respect to ratio, showing that it is possible in cases that the nondeterministic automaton admits some property (called t-twins). In [7], they also consider sum automata, providing an efficient determinization algorithm, which, however, is not guaranteed to be within a certain distance from the nondeterministic automaton. In general, we are unaware of any work on the approximation of automata over *infinite* words (such as limit-average and discounted-sum automata), nor of any work on approximate determinization with respect to arbitrary distance functions.

We investigate the potential of determinizing quantitative-automata approximately with respect to the difference and ratio functions (which are most commonly mentioned in the literature), as well as with respect to arbitrary ordered distance functions. We pay special attention to discounted-sum automata, whose approximate determinization yields the most promising results. We also study, in Section 2.4, the problem of approximate automata comparison and approximate game solving, showing how they can be solved by approximate automata determinization.

The, possibly, simplest quantitative automata value a run by the minimal/maximal weight along it, and they can be determinized exactly [8]. The next level of quantitative automata generalizes the Büchi and co-Büchi conditions, valuing a run by the maximal (LimSup) or minimal (LimInf) value that occurs infinitely often. LimInf automata can be determinized exactly, while LimSup cannot [8]. It turns out that LimSup automata cannot even be determinized approximately with respect to any distance function. Yet, they allow for exact determinization into a stronger automaton class, analogously to determinizing Büchi automata into Rabin or parity automata.

The more involved quantitative automata, which are inherently different from boolean automata, value a run by accumulating the weights along it. The basic accumulation schemes are either summation or averaging. Another aspect, on top of the accumulation, is the influence of time. With sum and average automata, all weights along the run have the same influence. In some cases, one favors close events over far away ones, thus introducing discounting of the weights along time. Discounted-sum automata are the best-known representatives of this class. On the other hand, there are cases in which one is only interested in the long-run behavior, looking at the limit values. The well-known representatives of this class are limit-average automata. Both discounted-summation and limit-average (also called mean-payoff) get intensive attention (e.g. [1, 3, 15]), and are even argued to be the canonic representatives of a major class of quantitative objectives [9].

We show that *sum* automata do not allow for an effective approximate determinization with respect to any ordered distance function. Furthermore, using the undecidability proof of their universality problem [2], we show that they cannot even be determinized approximately into any automaton class whose universality problem is decidable. On the positive side, restricting sum automata to *nonnegative* weights, while still not allowing for determinization, allows for approximate determinization with respect to some distance functions. We prove this by translating sum automata over nonnegative weights into product automata over weights in $[0, 1]$. The latter are shown to allow for approximate determinization with respect to the difference function.

Discounted-sum automata naturally allow for approximate determinization with respect to the difference function by unfolding the automaton computations up to a sufficient level. The smaller the required precision is, and the closer the discount-factor is to 1, the more expensive it is to determinize the automaton approximately. We represent the precision by $\varepsilon = 2^{-p}$ and the discount factor by $\lambda = 1 + 2^{-k}$. We analyze the unfolding approach to construct an automaton whose state space is exponential in the representation of the precision (p) and doubly exponential in the representation of the discount factor (k). We then provide an alternative construction that is singly exponential in k, in p, and in the number of states of the automaton. The construction generalizes the subset construction, keeping, for each state of the original automaton, its relative extra-cost ("gap"), within a fixed resolution. We complete the picture for discounted-sum automata by proving matching lower bounds, showing exponential dependency on each of these three parameters.

Average and *limit-average* automata use a more complicated accumulation scheme than summation. This gets an evident in approximate determinization, as they cannot be determinized approximately even if their weights are restricted to any pair of distinct values. It is left open whether there is a stronger automaton class with decidable properties into which average a limit-average automata can be determinized approximately.

Due to lack of space, the proofs are omitted in this version and can be found in the full version, at https://repository.ist.ac.at/102.

2 The Setting

We start with standard definitions of quantitative automata and distance functions, continue with our definition of approximate determinization, and conclude with describing how can one take advantage deterministic automata that approximate nondeterministic ones.

2.1 Quantitative Automata

A quantitative automaton is a weighted automaton, over finite or infinite words, valuing a run by a real number, and valuing a word by the infimum/supremum of the runs on it.

Formally, given an alphabet Σ, a *word* w over Σ is a finite or infinite sequence of letters in Σ, where $w[i]$ stands for the letter in position i, starting from 0. We denote the concatenation of a finite word u and a word w by $u{\cdot}w$, or simply by uw.

A *weighted automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_{in}, \delta, \gamma \rangle$ over a finite alphabet Σ, with a finite set of states Q, a subset of initial states $Q_{in} \subseteq Q$, a transition function $\delta \subseteq Q \times \Sigma \times Q$, and a weight function $\gamma : \delta \to \mathbb{Q}$. For a state q of $\mathcal{A}$, we denote by $\mathcal{A}^q$ the automaton that is identical to $\mathcal{A}$, except for having q as its initial state.

An automaton may have in general many possible transitions for each state and letter, and hence we say that $\mathcal{A}$ is *nondeterministic*. In the case where $|Q_{in}| = 1$ and for every $q \in Q$ and $\sigma \in \Sigma$, we have $|\{q' \mid (q, \sigma, q') \in \delta\}| \leq 1$, we say that $\mathcal{A}$ is *deterministic*. In the case where for every $q \in Q$ and $\sigma \in \Sigma$, we have $|\{q' \mid (q, \sigma, q') \in \delta\}| \geq 1$, we say that $\mathcal{A}$ is *complete*. Intuitively, a complete automaton cannot get stuck at some state. In this paper, we only consider complete automata.

A run of an automaton is a sequence of states and letters, $q_0, \sigma_1, q_1, \sigma_2, q_2, \ldots$, such that $q_0 \in Q_{in}$ and for every i, $(q_i, \sigma_{i+1}, q_{i+1}) \in \delta$. The length of a run, denoted $|r|$, is n for a finite run $r = q_0, \sigma_1, q_1, \ldots, \sigma_{n-1}, q_{n-1}$, and ∞ for an infinite run. A run r induces a sequence of weights, $\gamma_0, \gamma_1, \ldots$, where $\gamma_i = \gamma(q_i, \sigma_{i+1}, q_{i+1})$.

The value of a run r, denoted $\gamma(r)$, is a real number defined by a *value function* over its sequence of weights, depending on the automaton type. For an automaton $\mathcal{A}$, the value of a word w, denoted $\mathcal{A}(w)$, is either the infimum or the supremum of $\{\gamma(r) \mid r \text{ is a run of } \mathcal{A} \text{ on } w\}$, depending on $\mathcal{A}$'s type. A run r is *optimal* if $\gamma(r) = \mathcal{A}(w)$.

By the above definitions, an automaton $\mathcal{A}$ realizes a function from Σ^* or Σ^ω to $\mathbb{R}$. Two automata, $\mathcal{A}$ and $\mathcal{A}'$, are *equal* if they realize the same function.

Next, we define some types of quantitative automata, differing by their value function. We use below "inf of runs on w" as a shortcut for $\inf\{\gamma(r) \mid r \text{ is a run of } \mathcal{A} \text{ on } w\}$, and an analogous shortcut for the supremum.

Sum automata. Finite words; $\gamma(r) = \sum_{i=0}^{|r|-1} \gamma_i$; $\mathcal{A}(w) = \inf$ of runs on w.

Product automata. Finite words; $\gamma(r) = \prod_{i=0}^{|r|-1} \gamma_i$; $\mathcal{A}(w) = \sup$ of runs on w.

Average automata. Finite words; $\gamma(r) = \frac{1}{|r|} \sum_{i=0}^{|r|-1} \gamma_i$; $\mathcal{A}(w) = \sup$ of runs on w.

Discounted-sum automata. Operate over finite or infinite words, and are equipped with a rational discount factor $\lambda > 1$. [1] They value a run r by $\gamma(r) = \sum_{i=0}^{|r|-1} \frac{\gamma_i}{\lambda^i}$; and $\mathcal{A}(w) = \inf$ of runs on w. We write NDA and DDA to denote nondeterministic and deterministic discounted-sum automata, respectively, as well as λ-NDA or λ-DDA to denote an NDA or DDA with a discount factor λ, for example $\frac{5}{2}$-NDA.

LimSup Automata. Infinite words; $\gamma(r) = \lim_{n\to\infty} \sup\{\gamma_i \mid i \geq n\}$; $\mathcal{A}(w) = \sup$ of runs on w.

Limit-average Automata. Infinite words; $\gamma(r) = \lim_{n\to\infty} \inf\{\frac{1}{i} \sum_{j=0}^{i-1} \gamma_j \mid i \geq n\}$, [2] $\mathcal{A}(w) = \sup$ of runs on w.

[1] The discount factor λ is often used in the literature as a multiplying factor, rather than as a dividing factor, thus taking the role of $\frac{1}{\lambda}$, compared to our definitions.

[2] There is another version of limit-average automata, where $\gamma(r) = \lim_{n\to\infty} \sup\{\frac{1}{i} \sum_{j=0}^{i-1} \gamma_j \mid i \geq n\}$. All of our results equally apply to both versions.

2.2 Distance Functions

We restrict our attention to distance functions over $\mathbb{R}$, yet the definitions equally apply to every ordered space.

A function $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a *distance function* if for every $x, y, z \in \mathbb{R}$, it holds that:

1. $d(x, y) \geq 0$ (non-negative);
2. $d(x, y) = 0$ iff $x = y$ (identity of indiscernibles);
3. $d(x, y) = d(y, x)$ (symmetry); and
4. $d(x, y) + d(y, z) \geq d(x, z)$ (triangle inequality) .

We use distance functions for measuring the distance between two quantitative automata. Since these automata relate to the natural order on $\mathbb{R}$, using non-determinism for choosing the infimum or supremum of the run values, the relevant distance functions should also respect the order on $\mathbb{R}$. A distance function d is *ordered* if for every $x \leq y \leq z \in \mathbb{R}$, we have $d(x, y) \leq d(x, z)$ and $d(y, z) \leq d(x, z)$.

When we speak of the *difference function*, we refer to $d(x, y) = |x - y|$. Another common measure is *ratio*, however it is not a distance function, as it does not satisfy the triangle inequality. For that reason, when we speak of the *ratio distance function*, we refer to $d(x, y) = |\log x - \log y|$, which also measures the proportion between values.

2.3 Approximate Determinization

We define that an automaton can be determinized approximately if for every real precision $\varepsilon > 0$, there is a deterministic automaton such that the distance between their values on all words is less than or equal to ε. Formally,

▶ **Definition 1** (Approximation). The definitions below stand for approximation with respect to a distance function d.

- A quantitative automaton $\mathcal{A}'$ *ε-approximates* a quantitative automaton $\mathcal{A}$, for a real constant $\varepsilon > 0$, if for every word w, $d(\mathcal{A}(w), \mathcal{A}'(w)) \leq \varepsilon$.
- A quantitative automaton $\mathcal{A}$ can be *approximated* by an automaton class $\mathfrak{C}$ if for every real constant $\varepsilon > 0$ there is an automaton $\mathcal{A}' \in \mathfrak{C}$ that ε-approximates $\mathcal{A}$.
- A quantitative automaton $\mathcal{A}$ of a class $\mathfrak{C}$ can be *determinized approximately* if it can be approximated by the class of deterministic automata in $\mathfrak{C}$; the class $\mathfrak{C}$ can be *determinized approximately* if every automaton $\mathcal{A} \in \mathfrak{C}$ can.

2.4 Taking Advantage of Approximated Automata

Approximate determinization is useful for automata comparison, which is essential in formal verification, as well as for game solving, which is essential in synthesis.

Approximate automata comparison. Consider two nondeterministic quantitative automata $\mathcal{A}$ and $\mathcal{B}$. One can approximately solve the question of whether for all words w, $\mathcal{A}(w) \geq \mathcal{B}(w)$, with respect to the difference function and a precision $\varepsilon > 0$, by generating deterministic automata $\mathcal{A}'$ and $\mathcal{B}'$ that $\frac{\varepsilon}{2}$-approximate $\mathcal{A}$ and $\mathcal{B}$, respectively, and computing $m = \sup_w \{\mathcal{B}'(w) - \mathcal{A}'(w)\}$. If $m > \varepsilon$, it follows that $\mathcal{A}(w)$ is not always bigger than $\mathcal{B}(w)$. Otherwise, it follows that $\mathcal{A}(w)$ is always bigger than $\mathcal{B}(w)$, up to a possible mistake of 2ε. Equivalence and universality can be approximated similarly, up to any desired precision.

Note that one cannot solve the question of whether for all words w, $(\mathcal{B}(w) - \mathcal{A}(w)) \leq \varepsilon$, because it is as difficult as the question of whether $\mathcal{A} \geq \mathcal{B}$. Yet, one can reduce the uncertainty area to be arbitrarily small.

Approximate game solving. Consider a two-player game $\mathcal{G}$ whose winning condition is given by means of a nondeterministic quantitative automaton $\mathcal{A}$. For solving the game, one determinizes $\mathcal{A}$ into an automaton $\mathcal{D}$, takes the product of $\mathcal{G}$ and $\mathcal{D}$, and finds optimal strategies for the game $\mathcal{G}' = \mathcal{G} \times \mathcal{D}$. The value of the game is the value that $\mathcal{A}$ assigns to the trace of the game, once the two players follow their optimal strategies. Now, in the case that $\mathcal{D}$ is not equivalent to $\mathcal{A}$, but $\frac{\varepsilon}{2}$-approximates it, the value of $\mathcal{G}'$ is guaranteed to be up to ε-apart from the value of $\mathcal{G}$.

3 Sum Automata

Sum automata are the basic form of weighted automata that value a run by accumulating the weights along it. Having both positive and negative numbers allows for counting, which makes sum automata close enough, in some aspects, to counter machines. For that reason, their universality problem is undecidable [2]. Using this undecidability result, we show that sum automata cannot be effectively determinized approximately into any automaton class whose universality problem is decidable.

Restricting to *nonnegative* weights, the above undecidability result does not hold [2]. Indeed, we show that sum automata over nonnegative weights, while still not determinizable, allows for approximate determinization with respect to some distance functions.

We start with a lemma on the relation between sum and product automata.

▶ **Lemma 2.** *Consider a sum automaton $\mathcal{A}$, and construct from it a product automaton $\mathcal{B}$, by changing every weight x to 2^{-x}. Then for every word w, $\mathcal{B}(w) = 2^{-\mathcal{A}(w)}$.*

3.1 Unrestricted weights

We show that sum automata do not allow for an effective approximate determinization by using the undecidability proof of their universality problem.

▶ **Lemma 3** ([2]). *For every two-counter machine $\mathcal{M}$, there is a sum automaton $\mathcal{A}$ with weights in $\{-1, 0, 1\}$ such that M halts iff there is a word w such that $\mathcal{A}(w) > 0$.*

▶ **Theorem 4.** *Sum automata cannot be effectively determinized approximately with respect to any ordered distance function to any automaton class whose universality problem is decidable.*

Proof sketch. The key observation is that in order to solve the halting problem of two-counter machines, a deterministic automaton need not approximate the automaton $\mathcal{A}$ of Lemma 3 on all its possible values, but only on the values 0 and 1. Then, using the order property of the distance function, a close enough approximation will allow to deduce whether the accurate value is 0 or 1. ◀

As a special case of the above theorem, sum automata cannot be determinized approximately into deterministic sum automata, as their universality problem is equivalent to their emptiness problem, which is decidable (and polynomial).

By the relation between sum and product automata, we get the following corollary.

▶ **Corollary 5.** *Product automata over nonnegative weights cannot be effectively determinized approximately, with respect to any ordered distance function, to any automaton class whose universality problem is decidable.*

3.2 Nonnegative Weights

We show that sum automata over nonnegative weights cannot be determinized approximately with respect to difference, while they can be determinized approximately with respect to the distance function $d(x, y) = |2^{-x} - 2^{-y}|$. The usefulness of the latter distance function is arguable, yet it relates to two interesting observation. The first is the ability to determinize product automata approximately with respect to the natural difference function. The second is the conceptual difference between the sum and the average accumulation schemes, following from the inability to determinize approximately average automata even when restricting their weights (Section 5).

We start with the negative proof, using the standard sum automaton that computes the minimum between the number of a's and b's.

▶ **Theorem 6.** *Sum automata over nonnegative weights cannot be determinized approximately with respect to difference (even by sum automata with unrestricted weights).*

▶ Remark. Sum automata over positive weights cannot be determinized approximately with respect to ratio, using the same arguments as in the proof of Theorem 6.

▶ **Corollary 7.** *Product automata over weights in* $(0, 1]$ *cannot be determinized exactly nor determinized approximately with respect to ratio.*

We now turn to the positive results.

▶ **Theorem 8.** *Product automata over weights in* $[0, 1]$ *can be determinized approximately with respect to difference.*

▶ Remark. Product automata over weights in $[-1, 1]$ can also be determinized approximately with respect to difference. The proof of Theorem 8 does not hold, because it uses the monotonicity of $[0, 1]$-multiplications. Yet, one can properly extend the construction of the proof of Theorem 8 by keeping for each state both the maximal and the minimal accumulated values.

Using the relation between sum and product automata (Lemma 2), we get the following.

▶ **Theorem 9.** *Sum automata over nonnegative weights can be determinized approximately with respect to the distance function* $d(x, y) = |2^{-x} - 2^{-y}|$.

4 Discounted-Sum Automata

The naive approximation approach, of unfolding the automaton computations up to a sufficient level, is analyzed below to be doubly exponential in the representation of the discount factor. We then provide an alternative construction, based on keeping the relative extra-costs ("gaps") of the automaton states, and show that it is singly exponential in the representations of the precision, discount factor, and number of states. We conclude with proving matching lower bounds. In this section, when we speak of approximation, we mean approximation with respect to the difference function.

We start with the relation between discounted-sum automata over finite words and over infinite words, showing that approximation over finite words guarantees approximation over infinite words, but not vice versa.

▶ **Lemma 10.** *For every precision* $\varepsilon > 0$ *and discount factor* $\lambda > 1$*, if a* λ*-NDA* ε*-approximates another* λ*-NDA over finite words then it also* ε*-approximates it over infinite words. The converse need not hold.*

Following Lemma 10, it is enough to prove the correctness of the constructions with respect to finite words, and the lower bounds with respect to infinite words.

4.1 Approximation by Unfolding

We formalize below the naive approach of unfolding the automaton computation up to a sufficient level.

The Construction. Given an NDA $\mathcal{A} = \langle \Sigma, Q, Q_{in}, \delta, \gamma, \lambda \rangle$ and a parameter $l \in \mathbb{N}$, we construct a DDA $\mathcal{D}$ that is the depth-l unfolding of $\mathcal{A}$. We later fix the value of l to obtain a DDA that approximates $\mathcal{A}$ with a desired precision ε.

The DDA is $\mathcal{D} = \langle \Sigma, Q', q'_{in}, \delta', \gamma', \lambda \rangle$ where:

- $Q' = \Sigma^l$; the set of words of length l.
- q'_{in} = the empty word.
- $\delta' = \{(w, \sigma, w \cdot \sigma) \mid |w| \leq l - 1 \wedge \sigma \in \Sigma\} \cup \{(w, \sigma, w) \mid |w| = l \wedge \sigma \in \Sigma\}$.
- For all $w \in \Sigma^{\leq l-1}$, and $\sigma \in \Sigma$, let $\gamma'(w, \sigma, w \cdot \sigma) = (\mathcal{A}(w \cdot \sigma) - \mathcal{A}(w))/\lambda^{|w|}$; for all $w \in \Sigma^l$, and $\sigma \in \Sigma$, let $\gamma'(w, \sigma, w) = \frac{v+V}{2}$ where v and V are the smallest and largest weights in $\mathcal{A}$, respectively.

◀

The above construction yields an automaton whose state space might be doubly exponential in the representation of the discount factor.

▶ **Theorem 11.** *Consider a precision $\varepsilon = 2^{-p}$ and an NDA $\mathcal{A}$ with a discount factor $\lambda = 1 + 2^{-k}$ and maximal weight difference of m. Then applying the unfolding construction on $\mathcal{A}$, for a precision ε, generates a DDA $\mathcal{D}$ that ε-approximates $\mathcal{A}$ with up to $2^{\Theta(2^k(k+p+\log m))}$ states.*

4.2 Approximation by Gap Rounding

We start with defining the "cost" and the "gap" of a state, to be used in the determinization construction. The *cost* of reaching a state q of an NDA $\mathcal{A}$ over a finite word u is $\mathsf{cost}(q, u) = \min\{\gamma(r) \mid r \text{ is a run of } \mathcal{A} \text{ on } u \text{ ending in } q\}$, where $\min \emptyset = \infty$. The *gap* of q over a finite word u is $\mathsf{gap}(q, u) = \lambda^{|u|}(\mathsf{cost}(q, u) - \mathcal{A}(u))$. Note that when $\mathcal{A}$ operates over infinite words, we interpret $\mathcal{A}(u)$, for a finite word u, as if $\mathcal{A}$ was operating over finite words.

Intuitively, the gap of a state q over a word u stands for the weight that a run starting in q should save, compared to a run starting in u's optimal ending state, in order to make q's path optimal. A gap of a state q over a finite word u is said to be *recoverable* if there is a suffix that makes this path optimal; that is, if there is a word w, such that $\mathsf{cost}(q, u) + \frac{\mathcal{A}^q(w)}{\lambda^{|u|}} = \mathcal{A}(uw)$. The suffix w should be finite/infinite, depending on whether $\mathcal{A}$ operates over finite/infinite words.

The main idea in the determinization procedure of [6] is to extend the subset construction by keeping a recoverable-gap value to each element of the subset. It is shown [6] that for every integral factor the procedure is guaranteed to terminate, while for every non-integral rational factor it might not terminate. The problem with non-integral factors is that the recoverable-gaps might be arbitrarily dense, implying infinitely many gaps within the maximal bound of recoverable gaps.

Our approximation scheme generalizes the determinization procedure of [6], rounding the stored gaps to a fixed resolution. Since there is a bound on the maximal value of a recoverable gap, the fixed resolution guarantees the procedure's termination.

The question is, however, how an unbounded number of gap rounding allows for the required precision. The key observation is that the rounding is also discounted along the computation. For a λ-NDA, where $\lambda = 1 + 2^{-k}$, and a precision $\varepsilon = 2^{-p}$, we show that a resolution of $2^{-(p+k-1)}$ is sufficient. For an NDA whose maximal weight difference is m, the maximal recoverable gap is below $m2^{k+1}$. Hence, for an NDA with n states, the resulting DDA would have up to $2^{n(p+2k+\log m)}$ states.

The construction is formalized below, and illustrated with an example in Figure 1.)

The Construction. Consider a discount factor $\lambda = 1 + 2^{-k}$, with $k > 0$, and an NDA $\mathcal{A} = \langle \Sigma, Q = \langle q_1, \ldots, q_n \rangle, Q_{in}, \delta, \gamma, \lambda \rangle$, in which the maximal difference between the weights is m. For simplicity, we extend γ with $\gamma(\langle q_i, \sigma, q_j \rangle) = \infty$ for every $\langle q_i, \sigma, q_j \rangle \notin \delta$. Note that our discounted-sum automata do not have infinite weights; it is only used as an internal element of the construction.

For a precision $\varepsilon = 2^{-p}$, with $p > 0$, we construct a DDA $\mathcal{D} = \langle \Sigma, Q', q'_{in}, \delta', \gamma', \lambda \rangle$ that ε-approximates $\mathcal{A}$. We first define the set $G = \{i2^{-(p+k-1)} \mid i \in \mathbb{N} \text{ and } i \leq m2^{p+2k+1}\} \cup \{\infty\}$ of recoverable-gaps. The ∞ element denotes a non-recoverable gap, and behaves as the standard infinity element in the arithmetic operations that we will be using.

A state of $\mathcal{D}$ extends the standard subset construction by assigning a recoverable gap to each state of $\mathcal{A}$. That is, $Q' = \{\langle g_1, \ldots, g_n \rangle \mid \text{ for every } 1 \leq h \leq n, g_h \in G\}$.

The initial state of $\mathcal{D}$ is $q'_{in} = \langle g_1, \ldots, g_n \rangle$, where for every $1 \leq i \leq n$, $g_i = 0$ if $q_i \in Q_{in}$ and $g_i = \infty$ otherwise.

For bounding the number of possible states, the gaps are rounded to a resolution of $2^{-(p+k-1)}$. Formally, for every number $x \geq 0$, we define $\mathsf{Round}(x) = i2^{-(p+k-1)}$, such that $i \in \mathbb{N}$ and for every $j \in \mathbb{N}, |x - i2^{-(p+k-1)}| \leq |x - j2^{-(p+k-1)}|$.

For every state $q' = \langle g_1, \ldots, g_n \rangle \in Q'$ and letter $\sigma \in \Sigma$, we define the transition function $\delta(q', \sigma) = q'' = \langle x_1, \ldots, x_n \rangle$, and the weight function $\gamma(\langle q', \sigma, q'' \rangle) = c$ as follows.

- For every $1 \leq h \leq n$, we set $c_h := \min\{g_j + \gamma(\langle q_j, \sigma, q_h \rangle) \mid 1 \leq j \leq n\}$
- $c := \min_{1 \leq h \leq n} (c_h)$
- For every $1 \leq l \leq n$, $x_l := \mathsf{Round}(\lambda(c_h - c))$. if $x_l \geq m2^{k+1}$ then $x_l := \infty$

◀

The correctness and the state complexity of the construction is formalized below.

▶ **Theorem 12.** *Consider a discount factor $\lambda = 1 + 2^{-k}$ and a precision $\varepsilon = 2^{-p}$, for some positive numbers p and k. Then for every λ-NDA $\mathcal{A}$ with n states and weight difference of up to m, there is a λ-DDA that ε-approximates $\mathcal{A}$ with up to $2^{n(p+k+\log m)}$ states. The automata $\mathcal{A}$ and $\mathcal{D}$ may operate over finite words as well as over infinite words.*

Proof sketch. We show the correctness for finite words, implying, by Lemma 10, also the correctness for infinite words. We start by proving the claim with respect to an *infinite-state* automaton that is constructed as above, except for not changing any gap to ∞, and argue afterwards that changing all gaps that exceed $m2^{k+1}$ to ∞ does not harm the correctness.

We use the following notations: upon reading a word w, the constructed automaton yields the sequence $c_1, c_2, \ldots, c_{|w|}$ of weights, and reaches a state $\langle g_{1,w}, \ldots, g_{n,w} \rangle$. We denote half the gap resolution, namely $2^{-(p+k)}$, by r. Intuitively, $\frac{g_{h,w}}{\lambda^{|w|}} + \sum_{i=1}^{|w|} \frac{c_i}{\lambda^{i-1}}$ stands for the approximated cost of reaching the state q_h upon reading the word w. We define for every word w and $1 \leq h \leq n$, the "mistake" in the approximated cost by $M(h, w) = \frac{g_{h,w}}{\lambda^{|w|}} + \sum_{i=1}^{|w|} \frac{c_i}{\lambda^{i-1}} - \mathsf{cost}(q_h, w)$, and (delicately) show by induction on the length of w that $|M(h, w)| \leq \sum_{i=1}^{|w|} \frac{r}{\lambda^i}$.

◀

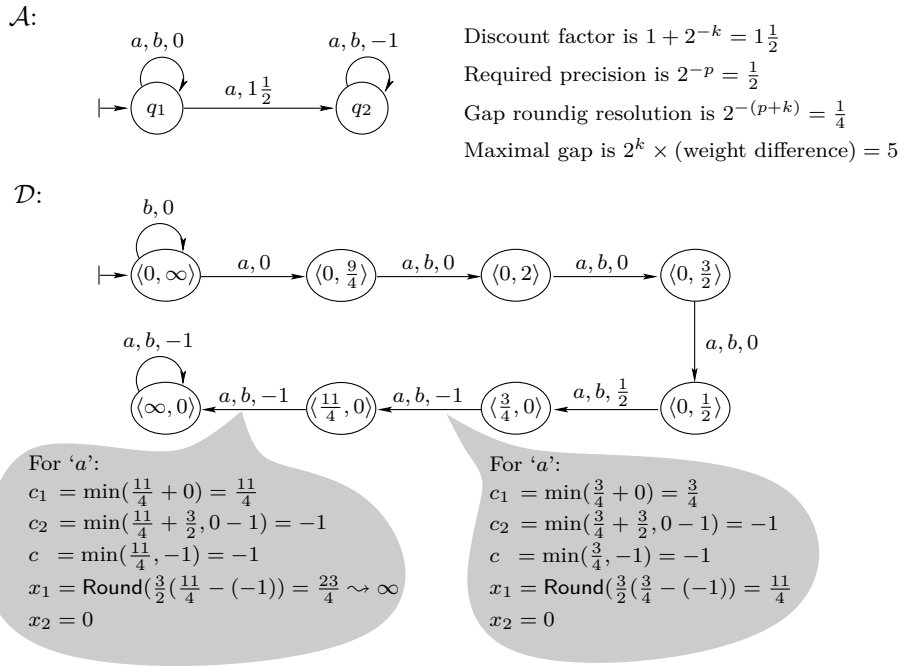


Figure 1 Determinizing the NDA $\mathcal{A}$ approximately into the DDA $\mathcal{D}$. The gray bubbles detail some of the intermediate calculations of the approximate-determinization construction.

4.3 Lower Bounds

The upper bound described in Section 4.2, for determinizing an NDA $\mathcal{A}$ approximately, exponentially depends on three parameters: n, denoting the number of states in $\mathcal{A}$; k, representing the proximity of $\mathcal{A}$'s discount factor to 1; and p, representing the precision. We show below that exponential dependency on these three factors is unavoidable.

For showing dependency on the precision (p), as well as on the discount factor (k), one can fix the other two parameters and show exponential dependency on the varying parameter. This is formalized in Theorems 13 and 14.

As for the number of states (n), there is no absolute dependency – one may approximate the non-deterministic automaton via the unfolding approach (Section 4.1), having no dependency on n. Yet, the trade off is a double-exponential dependency on k. Thus, one may check whether there is an exponential dependency on n, once p and k are fixed, and n remains below $O(2^k)$. This is indeed the case, as shown in Theorem 15.

▶ **Theorem 13.** *There is an NDA $\mathcal{A}$ with two states, such that for every $\varepsilon = 2^{-p} > 0$, every DDA (with any discount factor) that ε-approximates $\mathcal{A}$ has at least $2^{\lfloor \frac{p-2}{\log 3} \rfloor}$ states.*

▶ **Theorem 14.** *There is a family of NDAs, $\mathcal{A}_k$, for $k \geq 2$, with two states and discount factor $1 + 2^{-k}$ over an alphabet of two letters, such that every DDA (with any discount factor) that $\frac{1}{8}$-approximates $\mathcal{A}_k$ has at least 2^{k-1} states.*

▶ **Theorem 15.** *For every $k \geq 3$, there is a family of NDAs, $\mathcal{A}_n$, for $n \leq 2^k$, with $n + 1$ states and discount factor $1 + 2^{-k}$ over an alphabet of size $2n + 1$, such that every DDA (with any discount factor) that $\frac{1}{12}$-approximates $\mathcal{A}_n$ has at least 2^n states.*

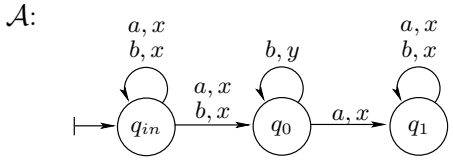


Figure 2 A limit-average automaton assigning y to words with finitely many a's and x otherwise.

5 Average and Limit-Average Automata

There are two main differences between summation and averaging, leading to different results and proofs regarding sum and average/limit-average automata. On the one hand, averaging adds a computation layer on top of summation, for which reason average and limit-average automata cannot be determinized approximately with respect to any distance function, even if restricting their weights. On the other hand, average and limit-average automata yield dense sets of values. Therefore, the undecidability of their universality problem does not imply that they cannot be effectively determinized approximately into *another* automaton class with decidable properties. The question of whether they can, or cannot, is left open.

5.1 Average Automata

Average automata are inherently different from sum automata by not falling into the class of automata based on a semi-ring (the average function does not have an identity element). Their universality problem is undecidable, directly following from the undecidability of the universality problem of sum automata, as for every sequence π of numbers, $Sum(\pi) \leq 0$ if and only if $Avg(\pi) \leq 0$. Yet, it does not imply that average automata cannot be determinized approximately – the corresponding proof that relates to sum automata (Theorem 4) relies on the fixed minimal distance between two values of a sum automaton, which is not the case with average automata. Nevertheless, a different proof is used to show that average automata cannot be determinized approximately.

▶ **Theorem 16.** *Average automata over any set of weights (with at least two distinct weights) cannot be determinized approximately with respect to any ordered distance function.*

5.2 Limit-Average Automata

As average automata, limit-average automata cannot be determinized approximately with respect to any distance function, and no weight restriction can overcome it. The proof is different from the one for average automata, providing a slightly stronger result, namely that limit-average automata cannot be determinized approximately even with respect to unordered distance functions.

▶ **Theorem 17.** *Limit-average automata over any set of weights (with at least two distinct weights) cannot be determinized approximately with respect to any distance function.*

Proof sketch. We use the automaton $\mathcal{A}$, defined in Figure 2, with arbitrary weights $x < y$. Considering a deterministic automaton that properly approximates it, we analyze the possible average weights of its cycles along words with only a's and only b's. We then construct a corresponding word in which the frequency of a's decays exponentially, and show that the deterministic automaton misvalues it. ◀

6 Conclusions

Automata comparison and game solving are critical tasks in formal verification and synthesis. Both require, or are closely related to, automata determinization. Quantitative automata play a key role in the emerging area of quantitative formal methods, but, unfortunately, they usually cannot be determinized. For that reason, quantitative verification and synthesis are generally undecidable. It is thus natural to ask for approximate solutions, based on approximate automata determinization. Moreover, having automata with real values, rather than boolean ones, brings a natural potential for such approximations.

With discounted-sum automata, we gave a construction fulfilling this potential. It can be used for systems with inherent discounting, as well as for other systems, if willing to introduce some discounting. With automata that are based on an accumulation such as sum or product, weight restriction allows for approximate determinization with respect to some distance functions. On the other hand, automata that are based on averaging or limit-averaging cannot be determinized approximately with respect to any distance function, and no weight restriction (with at least two different weights) can overcome this.

Acknowledgment. We thank Laurent Doyen for great ideas and valuable help in analyzing discounted-sum automata.

References

1 L. de Alfaro, T. A. Henzinger, and R. Majumdar. Discounting the future in systems theory. In *Proc. 30th ICALP conference*, pages 1022–1037, 2003.

2 S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *Proc. of ATVA11*, pages 482–491, 2011.

3 R. Alur and A. Trivedi. Relating average and discounted costs for quantitative analysis of timed systems. In *Proc. 11th EMSOFT conference*, pages 165–174, 2011.

4 B. Aminof, O. Kupferman, and R. Lampert. Rigorous approximated determinization of weighted automata. In *Proc. 26th LICS symposium*, pages 345–354, 2011.

5 R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. 21st CAV conference*, pages 140–156, 2009.

6 U. Boker and T. A. Henzinger. Determinizing discounted-sum automata. In *Proc. 20th Annual Conf. of the European Association for Computer Science Logic*, 2011.

7 A. L. Buchsbaum, R. Giancarlo, and J. Westbrook. An approximate determinization algorithm for weighted finite-state automata. *Algorithmica*, 30(4):503–526, 2001.

8 K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. In *Proc. of CSL*, LNCS 5213, pages 385–400. Springer, 2008.

9 K. Chatterjee, L. Doyen, and R. Singh. On memoryless quantitative objectives. In *Proc. 18th FCT symposium*, pages 148–159, 2011.

10 K. Chatterjee, M. Randour, and J.F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In *Proc. 23rd CONCUR conference*, 2012.

11 A. Degorre, L. Doyen, R. Gentilini, J. F. Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In *Proc. of CSL*, pages 260–274, 2010.

12 M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 2009.

13 T. A. Henzinger and N. Piterman. Solving games without determinization. In *proc. of 15th EACSL conf.*, pages 395–410, 2006.

14 O. Kupferman and Y. Lustig. Lattice automata. In *8th VMCAI conf.*, pages 199–213, 2007.

15 U. Zwick and M.S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

Timed Lossy Channel Systems*

Parosh Aziz Abdulla[1], Mohamed Faouzi Atig[1], and Jonathan Cederberg[1]

1 Uppsala University, Sweden

Abstract

Lossy channel systems are a classical model with applications ranging from the modeling of communication protocols to programs running on weak memory models. All existing work assume that messages traveling inside the channels are picked from a finite alphabet. In this paper, we extend the model by assuming that each message is equipped with a clock representing the age of the message, thus obtaining the model of *Timed Lossy Channel Systems (TLCS)*. The main contribution of the paper is to show that the control state reachability problem is decidable for TLCS.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Lossy channel systems, timed automata, model checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.374

1 Introduction

During the last two decades there has been a large amount of work devoted to the verification of *discrete* program models that have *infinite* state spaces such as Petri nets, pushdown systems, counter automata, and channel machines. In particular *lossy channel systems* have been studied extensively as a model of communication protocols. Such protocols are designed to work correctly even in the case where the underlying medium is unreliable in the sense that it can lose messages [8]. Recently, lossy channel systems have been proposed as a fundamental tool for describing programs running on *weak memories* [10, 6] since they are able to capture the behaviors of classical models such as TSO and PSO. In parallel, *timed automata* [9, 15, 14] are the most widely used model for the analysis of systems with *timed* behaviors. Several works have augmented discrete infinite-state models with timed behaviors. For instance, many different formalisms have been proposed for extending Petri nets with clocks and timed constraints, leading to various definitions of *Timed Petri Nets* (e.g., [12, 5]). Also, several works [4, 13, 11, 17, 18, 19, 22] consider timed pushdown automata. In this paper, we consider *(Dense-)Timed Lossy Channel Systems* (or *TLCS* for short). A TLCS combines the classical models of lossy channel systems and timed automata. More precisely, a TLCS consists of finite number of processes. The processes operate on finite set of real-valued clocks, together with a finite number of lossy channels each of which behaves as an unbounded FIFO buffer. Each message traveling inside a channel is equipped with a real-valued clock representing its "age". Processes can *send* messages to the channels in which case the message is appended to the end of the channel. A *receive* operation may only take place if the message at the head of the channel is of the correct type and only if its age lies in a pre-defined interval associated with the transition. In a similar manner to timed automata, a transition may be conditioned by the values of the clocks. In a *timed transition*,

* Supported by the Swedish Research Council within the UPMARC Linnaeus centre of Excellence

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 374–386

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

the clock values and the ages of all the messages inside the channels are increased uniformly (by the same real number). Finally, any message inside a channel may non-deterministically be lost (deleted from the channel). The TLCS model thus subsumes both the models of lossy channel systems and timed automata. More precisely, we obtain the former if we prevent the TLCS from using the timed information (all the timing constraints are trivially valid); and obtain the latter if we prevent the TLCS from using the channels (no symbols are sent or received from the channels). Notice that a TLCS induces a system that is infinite in two dimensions, namely it has channels containing unbounded numbers of messages, and each message is equipped with a real-valued clock.

In this paper, we show decidability of the control state reachability problem for TLCS. We show the decidability result through a novel reduction formulated in two steps. First, we introduce a new model called *Dynamic Lossy Channel Systems (DLCS)* which is a generalization of (untimed) LCS. More precisely, a DLCS contains, in addition to a (fixed) finite set of lossy channels, a *dynamic* part that contains an *a priori* unbounded number of channels. The dynamic part behaves as a *second-order* lossy channel, i.e., a "lossy channel of lossy channels". We show that each DLCS induces a transition system that is *well quasi-ordered* in the sense of [7, 1], and thus the control state reachability problem is decidable for DLCS. In the second step, we reduce the control state reachability problem for TLCS to the the control state reachability problem for DLCS and thus prove the decidability of the former.

The complexity of the reachability problem for TLCS is not primitive recursive as it is not primitive recursive already for untimed LCS [16].

2 Preliminaries

Notation

We use $\mathbb{N}$ and $\mathbb{R}^{\geq 0}$ to denote the sets of natural numbers resp. non-negative reals. For a real number $r \in \mathbb{R}^{\geq 0}$, we define $\mathsf{Int}(r)$ as the greatest $n \in \mathbb{N}$ such that $n \leq r$, and $\mathsf{Frac}(r)$ as $r - \mathsf{Int}(r)$. We call $\mathsf{Int}(r)$ the *integer part* and $\mathsf{Frac}(r)$ the *fractional part* of r respectively. An open interval is written as (i, j) where $i \in \mathbb{N}$ and $j \in \mathbb{N} \cup \{\infty\}$. Intervals can also be closed in one or both directions, e.g. $[i, j]$ is closed in both directions and $[i, j)$ is closed to the left and open to the right. We denote the set of all intervals by $\mathcal{I}$. For $n \in \mathbb{N}$, we define the set $[n]^0 := \{0, 1, \ldots, n\}$, and define $[n]^1 := \{1, 2, \ldots, n\}$. For sets A and B, we use $h : A \to B$ to denote that h is a total function from A to B, and use $h[a \mapsto b]$ to denote the function h' where $h'(a) = b$ and $h'(a') = h(a')$ if $a' \neq a$. We use $(A \to B)$ to denote the set of total functions from A to B. We say that a function $f : \mathbb{N} \to \mathbb{N}$ is *strictly increasing* if whenever $i < j$ we also have $f(i) < f(j)$. We use A^* to denote the set of finite words over A. For words $w_1, w_2 \in A^*$, we use $w_1 \cdot w_2$ to denote the concatenation of w_1 and w_2. We use ϵ to denote the empty word. For a word $w = a_1 \cdots a_n$, we use $w[i]$ to denote the ith symbol a_i in w, and we will write $a \in w$ if $a = w[i]$ for some $i : 1 \leq i \leq n$. We will use a similar notation for tuples. We recall the classical subword ordering $\sqsubseteq$ on the set A^* of words, where $a_1 \ldots a_m \sqsubseteq a'_1 \cdots a'_n$ if there is a strictly increasing injection $g : [m]^1 \to [n]^1$ such that $a_i = a'_{g(i)}$. To simplify the notation, we write $\omega \in (A^*)^*$ as $\langle w_1 \rangle \cdots \langle w_n \rangle$ where $w_1, \cdots, w_n$ are words in A^*. We extend the ordering $\sqsubseteq$ to $(A^*)^*$ in such a way that $\omega = \langle w_1 \rangle \cdots \langle w_n \rangle \sqsubseteq \langle w'_1 \rangle \cdots \langle w'_n \rangle = \omega'$ if there is a strictly increasing injection $h : [m]^1 \to [n]^1$ where $w_i \sqsubseteq w'_{h(i)}$.

Transition Systems

A *transition system* is a pair $\mathcal{S} = \langle \Gamma, \longrightarrow \rangle$ where Γ is the set of configurations, and $\longrightarrow \subseteq \Gamma \times \Gamma$ is a binary relation on the set of configurations. As usual, we write $\gamma_1 \longrightarrow \gamma_2$ instead of $\langle \gamma_1, \gamma_2 \rangle \in \longrightarrow$. We use $\xrightarrow{*}$ to denote the reflexive transitive closure of $\longrightarrow$. For a set $\Gamma' \subseteq \Gamma$ of configurations, we define the set $Pre\,(\Gamma') := \{\gamma |\ \exists \gamma' \in \Gamma'.\, \gamma \longrightarrow \gamma'\}$. Sometimes, we equip the set Γ with an ordering $\trianglelefteq$ and write the transition system as a triple $\langle \Gamma, \longrightarrow, \trianglelefteq \rangle$. We say that $\mathcal{S}$ is *monotone* (wrt. $\trianglelefteq$) if whenever $\gamma_1 \longrightarrow \gamma_2$ and $\gamma_1 \trianglelefteq \gamma_3$ then $\gamma_2 \xrightarrow{*} \gamma_4$ for some γ_4 with $\gamma_3 \trianglelefteq \gamma_4$. We say that $\trianglelefteq$ is a *well quasi-ordering* (*wqo* for short), if, for all sequences $\gamma_0, \gamma_1, \gamma_2, \ldots$, there are $i < j$ with $\gamma_i \trianglelefteq \gamma_j$. A set $U \subseteq \Gamma$ is *upward closed* if whenever $\gamma_1 \in U$ and $\gamma_1 \trianglelefteq \gamma_2$ then $\gamma_2 \in U$. The upward closure of a set $\Gamma' \subseteq \Gamma$ is defined by $\Gamma'{\uparrow} := \{\gamma \in \Gamma |\ \exists d \in \Gamma'.\, d \trianglelefteq \gamma\}$. For sets $\Gamma'_1 \subseteq \Gamma'_2 \subseteq \Gamma$, we say that Γ'_1 is a *minor* of Γ'_2 if (i) for each $\gamma_2 \in \Gamma'_2$ there is a $\gamma_1 \in \Gamma'_1$ such that $\gamma_1 \trianglelefteq \gamma_2$, and (ii) $\gamma_1 \trianglelefteq \gamma_2$ implies $\gamma_1 = \gamma_2$ for all $\gamma_1, \gamma_2 \in \Gamma'_1$. If $\trianglelefteq$ is a wqo, then each minor is finite. However, in general, a set may have several different minors. In the applications of this paper, each set Γ' has a unique minor, denoted $\min(\Gamma')$. An instance of the *coverability problem* consists of two configurations γ_1 and γ_2. The task is to check whether $\gamma_1 \xrightarrow{*} \gamma_2{\uparrow}$. A transition system $\langle \Gamma, \longrightarrow, \trianglelefteq \rangle$ is said to be *well quasi-ordered* if the following conditions are satisfied: (i) $\trianglelefteq$ is computable, i.e., for given configurations γ, γ', we can check whether $\gamma_1 \trianglelefteq \gamma'$, (ii) $\trianglelefteq$ is a wqo, (iii) $\longrightarrow$ is monotone wrt. $\trianglelefteq$, (iv) for a configuration γ, we can compute the (finite) set $\min\,(Pre\,(\{\gamma\}{\uparrow}))$. Notice that, since the transition relation is monotone with respect to $\trianglelefteq$, it follows that the set $Pre\,(\{\gamma\}{\uparrow})$ is upward closed. The classical framework of well quasi-ordered transition systems [7, 1] provides the following sufficient conditions for decidability of the coverability problem.

▶ **Theorem 1.** *The coverability problem is decidable for well quasi-ordered transition systems.*

3 Timed Lossy Channel Systems

In this section, we introduce TLCS, define their operational semantics, and present the reachability problem. Furthermore, we show that it is sufficient to consider a class of "normalized" TLCS where initial ages of messages and values assigned to clocks are always 0.

A TLCS has three parts, a control part, a finite set of clocks, and a finite set of channels. The control part is a finite-state labeled transition system, where the labels are either clock operations or channel operations. The control part can be used to model the total behavior of a number of processes that communicate through the channels. The clocks assume real values, while the channels are unbounded lossy FIFO buffers.

Model

A *Timed Lossy Channel System* (*TLCS* for short) is a tuple $\mathcal{T} = \langle S, s_{init}, C, M, X, \Delta \rangle$, where S is a finite set of (control) states, $s_{init} \in S$ is the initial control state, C is a finite set of channels, M is a finite set of messages, X is a finite set of clocks, and Δ is a finite set of transitions. A transition $t \in \Delta$ is a triple $\langle s_1, op, s_2 \rangle$ where $s_1, s_2 \in S$ are states and op is an operation of one of the following forms:

1. nop is an empty operation that does not check or update the clock values or the channel contents.

2. $c!(m \in I)$ appends a new message $m \in M$ to the end of the channel $c \in C$. The initial age of the new message is selected non-deterministically from $I \in \mathcal{I}$.
3. $c?(m \in I)$ removes (receives) the message at the head of the channel $c \in C$ provided that this message is $m \in M$ and that its age lies in $I \in \mathcal{I}$.
4. $x \in I$ checks whether the value of $x \in X$ belongs to the interval $I \in \mathcal{I}$.
5. $x \leftarrow I$ assigns non-deterministically a value to $x \in X$ from $I \in \mathcal{I}$.

Configurations

A configuration γ of $\mathcal{T}$ is a triple $\langle s, \mathtt{X}, \nu \rangle$, where $s \in S$ is a control state, $\mathtt{X} \in \left(X \to \mathbb{R}^{\geq 0}\right)$ defines the clock values (assigns a real number to each clock), and $\nu \in \left(C \to (M \times \mathbb{R}^{\geq 0})^*\right)$ defines the content of each channel (the content of a channel is represented by a word, where each message is represented by a pair containing its name and its age).

Transition Relation

We define a transition relation on configurations $\longrightarrow_{\mathcal{T}} := \xrightarrow{D}_{\mathcal{T}} \cup \xrightarrow{T}_{\mathcal{T}} \cup \xrightarrow{\mathcal{L}}_{\mathcal{T}}$ as the union of a discrete transition relation $\xrightarrow{D}_{\mathcal{T}}$, a timed transition relation $\xrightarrow{T}_{\mathcal{T}}$, and a lossy transition relation $\xrightarrow{\mathcal{L}}_{\mathcal{T}}$.

We define the discrete transition relation as the union $\xrightarrow{D}_{\mathcal{T}} := \bigcup_{t \in \Delta} \xrightarrow{t}_{\mathcal{T}}$ of the transition relations induced by all transitions in Δ. For configurations $\gamma_1 = \langle s_1, \mathtt{X}_1, \nu_1 \rangle$, $\gamma_2 = \langle s_2, \mathtt{X}_2, \nu_2 \rangle$, and a transition $t = \langle s_1, op, s_2 \rangle \in \Delta$, we have $\gamma_1 \xrightarrow{t} \gamma_2$ if one of the following conditions holds:

1. $op = \mathsf{nop}$, $\mathtt{X}_2 = \mathtt{X}_1$, and $\nu_2 = \nu_1$. The empty operation does not affect the clock values or the channel contents.
2. $op = c!(m \in I)$, $\mathtt{X}_2 = \mathtt{X}_1$, $\nu_2 = \nu_1[c \mapsto (m, \delta) \cdot \nu_1(c)]$, and $\delta \in I$. The transition appends a new message to the end of the channel c with name m, and with an age that belongs to the interval I.
3. $op = c?(m \in I)$, $\mathtt{X}_2 = \mathtt{X}_1$, $\nu_1 = \nu_2[c \mapsto \nu_2(c) \cdot (m, \delta)]$, and $\delta \in I$. The transition removes the message at the head of the channel c provided that its name is m, and that its age is in the interval I.
4. $op = x \in I$, $\mathtt{X}_1(x) \in I$, $\mathtt{X}_2 = \mathtt{X}_1$, and $\nu_2 = \nu_1$. The transition is enabled only if the value of x belongs to I. The clock values and the channel contents are not affected.
5. $op = x \leftarrow I$, $\mathtt{X}_2 = \mathtt{X}_1[x \mapsto \delta]$, $\delta \in I$, and $\nu_2 = \nu_1$. The transition assigns a new value (belonging to I) to the clock x.

Notice that in all five cases the control state changes from s_1 to s_2.

The timed transition relation models the passage of time, in the sense that the values of all clocks and the ages of all messages inside the channels are uniformly increased by (the same) real number. For configurations $\gamma_1 = \langle s, \mathtt{X}_1, \nu_1 \rangle$, $\gamma_2 = \langle s, \mathtt{X}_2, \nu_2 \rangle$, and a real number $\delta \in \mathbb{R}^{\geq 0}$, the relation $\gamma_1 \xrightarrow{\delta}_{\mathcal{T}} \gamma_2$ holds if the following two conditions hold: (i) $\mathtt{X}_2(x) = \mathtt{X}_1(x) + \delta$ for all $x \in X$, and (ii) for every $c \in C$, if $\nu_1(c)$ is of the form $(m_1, \delta_1) \cdots (m_n, \delta_n)$ then ν_2 is of the form $(m_1, \delta_1 + \delta) \cdots (m_n, \delta_n + \delta)$. We write $\gamma_1 \xrightarrow{T}_{\mathcal{T}} \gamma_2$ to denote that $\gamma_1 \xrightarrow{\delta}_{\mathcal{T}} \gamma_2$ for some $\delta \in \mathbb{R}^{\geq 0}$.

Finally the lossy transition relation allows messages to be lost from the channels at any time. Formally, if $\gamma_1 = \langle s, \mathtt{X}, \nu_1 \rangle$ and $\gamma_2 = \langle s, \mathtt{X}, \nu_2 \rangle$, the relation $\gamma_1 \xrightarrow{\mathcal{L}}_{\mathcal{T}} \gamma_2$ holds if $\nu_2(c) \sqsubseteq \nu_1(c)$ for all $c \in C$.

Reachability

The initial configuration of a TLCS $\mathcal{T}$ is defined by $\gamma_{init} := \langle s_{init}, \mathtt{X}_{init}, \nu_{init} \rangle$ where $\mathtt{X}_{init}(x) = 0$ for all $x \in X$, and $\nu_{init}(c) = \epsilon$ for all $c \in C$. In other words, $\mathcal{T}$ is initiated from a configuration where it is in its initial control state, where all the clocks have a value equal to 0, and where all the channels are empty. A control state $s \in S$ is said to be *reachable* if $\gamma_{init} \xrightarrow{*}_{\mathcal{T}} \langle s, \mathtt{X}, \nu \rangle$ for some $\mathtt{X}$ and ν. An instance of the reachability problem consists of an TLCS $\mathcal{T} = \langle S, s_{init}, C, M, X, \Delta \rangle$ and a control state $s \in S$. The task is to check whether s is reachable.

Normalization

A TLCS $\mathcal{T} = \langle S, s_{init}, C, M, X, \Delta \rangle$ such that $I = [0, 0]$ for all $\langle s_1, c!(m \in I), s_2 \rangle \in \Delta$ is said to be *message-normalized*. We say that $\mathcal{T}$ is *clock-normalized* if whenever $\langle s_1, x \leftarrow I, s_2 \rangle \in \Delta$ then $I = [0, 0]$. Finally, $\mathcal{T}$ is *normalized* if it is both clock- and message-normalized. The following two lemmas show that the reachability problem for general TLCS can be reduced to that for normalized TLCS. Therefore, in the rest of the paper, we assume that all TLCS are normalized.

▶ **Lemma 2.** *The reachability problem for TLCS can be reduced to the reachability problem for message-normalized TLCS.*

▶ **Lemma 3.** *The reachability problem for TLCS can be reduced to the reachability problem for clock-normalized TLCS.*

4 Dynamic Lossy Channel Systems

In this section, we introduce the model of Dynamic Lossy Channel Systems (*DLCS* for short). The model is a generalization of lossy channel systems [8] in the sense that it contains a second-order channel (a "channel of channels"). A DLCS consists of three parts: a control part, a static part, and a dynamic part. The control part is a finite-state labeled transition system. The static part consists of a finite set of (static) channels, each of which contains a sequence of messages from a finite alphabet. The dynamic part contains a (possibly unbounded) sequence of (dynamic) channels over the same alphabet. Each transition of the control part may be labeled by an operation on the static or dynamic channels. In the former case, the operation may remove a message from the head of a static channel or insert a message at its end (as in the case of lossy channels). In the latter case, the operation may copy the content of a static channel and append it (as a new channel) to the end of the sequence of dynamic channels (thus creating a new channel at the leftmost position of the dynamic part), or copy the content of the rightmost dynamic channel (the one at the head of the sequence of channels) to a static channel and then delete this dynamic channel. Furthermore, messages inside any channel can be lost (deleted) non-deterministically, and also any (whole) dynamic channel may be lost non-deterministically. The static channels are static (they can cannot be created, deleted, or lost). Notice that all the channels in the system are unbounded and that there is no bound on the number of dynamic channels that may be created during a run of the system.

Model

A *DLCS* is a tuple $\mathcal{D} = \langle S, s_{init}, C, \Sigma, \Delta \rangle$ where S is a finite set of (control) states, $s_{init} \in S$ is the initial control state, C is a finite set of channels names, Σ is the channel alphabet,

and Δ is a finite set of transitions. A transition $t \in \Delta$ is a triple $\langle s_1, op, s_2 \rangle$ where $s_1, s_2 \in S$ are states and op is an operation of one of the following forms:

1. nop is an empty operation that does not check or update the channels,
2. $c!m$ appends the message $m \in \Sigma$ to the end of the static channel $c \in C$,
3. $c?m$ removes the message $m \in \Sigma$ from the head of the static channel $c \in C$,
4. $send\_channel(c)$ makes a copy of the content of the static channel c to a new dynamic channel, and appends the new channel to the end of the sequence of dynamic channels.
5. $receive\_channel(c)$ copies the content of the rightmost dynamic channel to the static channel $c \in C$ and then removes this dynamic channel from the sequence of channels.

Configurations

A configuration d of $\mathcal{D}$ is a triple $\langle s, \nu, \omega \rangle$, where $s \in S$ is a control state, $\nu \in (C \to \Sigma^*)$ is a function that represents the content of the set of static channels C, and $\omega \in (\Sigma^*)^*$ is the content of the sequence of dynamic channels, also called the dynamic part of $\mathcal{D}$.

For configurations $d_1 = \langle s_1, \nu_1, \omega_1 \rangle$, $d_2 = \langle s_2, \nu_2, \omega_2 \rangle$, we say that $d_1 \sqsubseteq d_2$ if $s_1 = s_2$, $\nu_1(c) \sqsubseteq \nu_2(c)$ for all $c \in C$, and $\omega_1 \sqsubseteq \omega_2$ (recall the definition of $\sqsubseteq$ from Section 2). Intuitively, we derive d_1 from d_2 by deleting messages from the channels (both static and dynamic) and by removing dynamic channels.

Transition Relation

We define the transition relation as the set $\longrightarrow_{\mathcal{D}} := \left(\bigcup_{t \in \Delta} \xrightarrow{t}_{\mathcal{D}}\right) \cup \xrightarrow{\mathcal{L}}_{\mathcal{D}}$ where $\bigcup_{t \in \Delta} \xrightarrow{t}_{\mathcal{D}}$ is the union of transition relations induced by all transitions in Δ, and $d_1 \xrightarrow{\mathcal{L}}_{\mathcal{D}} d_2$ whenever $d_2 \sqsubseteq d_1$. The relation $\xrightarrow{\mathcal{L}}_{\mathcal{D}}$ models the loss of messages and dynamic channels. For configurations $d_1 = \langle s_1, \nu_1, \omega_1 \rangle$, $d_2 = \langle s_2, \nu_2, \omega_2 \rangle$, and a transition $t = \langle s_1, op, s_2 \rangle \in \Delta$, we have $d_1 \xrightarrow{t}_{\mathcal{D}} d_2$ if one of the following conditions holds:

1. $op =$ nop, $\nu_1 = \nu_2$, and $\omega_1 = \omega_2$.
2. $c!m$, $\nu_2 = \nu_1[c \mapsto m \cdot \nu_1(c)]$, and $\omega_2 = \omega_1$. The message m is appended to the end of the channel c.
3. $c?m$, $\nu_1 = \nu_2[c \mapsto \nu_2(c) \cdot m]$, and $\omega_2 = \omega_1$. The message m is received (deleted) from the head of the channel c.
4. $send\_channel(c)$, $\nu_1 = \nu_2$, and $\omega_2 = \langle \nu_1(c) \rangle \cdot \omega_1$. A copy of the content of the static channel c is appended (as a new channel) to the end of the dynamic part of $\mathcal{D}$.
5. $receive\_channel(c)$, $\nu_2 = \nu_1[c \mapsto w]$, and $\omega_1 = \omega_2 \cdot \langle w \rangle$. The content of the right-most dynamic channel is copied to the static channel $c \in C$. The right-most dynamic channel is then removed.

Reachability

The initial configuration of an DLCS $\mathcal{D}$ is defined by $d_{init} := \langle s_{init}, \nu_{init}, \omega_{init} \rangle$ where $\nu_{init}(c) = \epsilon$ for all $c \in C$, and $\omega_{init} = \epsilon$. In other words, $\mathcal{D}$ is initiated from a configuration where it is in its initial control state, all the static channels are empty, and the sequence of dynamic channels is empty (no channel has yet been appended). We define the control state reachability problem (or simply the reachability problem in the sequel) in a similar manner to the case of TLCS (cf. Section 3). Notice that the checking of reachability of a control state s can translated to the coverability problem $d_{init} \xrightarrow{*}_{\mathcal{D}} \langle s, \nu_{init}, \omega_{init} \rangle \uparrow$.

▶ **Lemma 4.** *Any transition system* $\langle \Gamma, \longrightarrow, \sqsubseteq \rangle$ *induced by a DLCS is well quasi-ordered.*

Proof. We prove the lemma by showing that each of the four conditions in the definition of well quasi-ordered transition systems given in Section 2 holds.

1. The ordering defined is clearly computable.
2. Since any finite set is well quasi-ordered and also tuples and words over well quasi-ordered sets are well quasi-ordered [21], the ordering $\sqsubseteq$ as defined on configurations is a well quasi-ordering.
3. Assume $d_1 \longrightarrow d_2$ and $d_1 \sqsubseteq d_3$. From the definition of $\longrightarrow$, we get that $d_3 \xrightarrow{\mathcal{L}} d_1$, and by transitivity we immediately get $d_3 \xrightarrow{*} d_2$. Thus, $\longrightarrow$ is monotone wrt. $\sqsubseteq$.
4. Assume a configuration $d = \langle s, \nu, \omega \rangle$. We define $\min(Pre(\{d\}\uparrow)) := \min\left(\bigcup_{t \in \Delta} \min(Pre(t)(\{d\}\uparrow)) \cup \{d\}\right)$, where $Pre(t)(\{d\}\uparrow) = \{d_1 \mid \exists d_2 \in \{d\}\uparrow \,.\, d_1 \xrightarrow{t}_{\mathcal{D}} d_2\}$ is the predecessor relation wrt. the transition $t \in \Delta$. Consider a transition $t = \langle s_1, op, s_2 \rangle \in \Delta$. We define $\min(Pre(t)(\{d\}\uparrow))$ as a set A with the following properties. If $s \neq s_2$ then $A := \emptyset$. Otherwise, we have:
   - If $op = \mathsf{nop}$ then $A = \{\langle s_1, \nu, \omega \rangle\}$.
   - If $op = c!m$ and $\nu(c)$ is of the form $m \cdot w$ then $A := \{\langle s_1, \nu[c \mapsto w], \omega \rangle\}$.
   - If $op = c!m$, $\nu(c)$ is of the form $m' \cdot w$, and $m' \neq m$, then $A := \{\langle s_1, \nu, \omega \rangle\}$.
   - If $op = c?m$ then $A := \{\langle s_1, \nu[c \mapsto w \cdot m], \omega \rangle\}$.
   - If $op = send\_channel(c)$ and ω is of the form $\langle w \rangle \cdot \omega'$ then $A := \min(\{\langle s_1, \nu[c \mapsto w'], \omega \rangle \mid (\nu(c) \sqsubseteq w') \wedge (w \sqsubseteq w')\} \cup \{\langle s_1, \nu, \omega \rangle\})$
   - If $op = send\_channel(c)$ and $\omega = \epsilon$ then $A := \{\langle s_1, \nu, \omega \rangle\}$.
   - If $op = receive\_channel(c)$ then $A := \{\langle s_1, \nu[c \mapsto \epsilon], \omega \cdot \langle \nu(c) \rangle \rangle\}$.

◀

From this and Theorem 1 we get the following theorem.

▶ **Theorem 5.** *The reachability problem is decidable for DLCS.*

5 From TLCS to DLCS

In this section, we show how we can encode a TLCS by a DLCS such that we preserve control state reachability. This enables us to extend decidability of the reachability problem from DLCS to TLCS.

▶ **Theorem 6.** *The reachability problem is decidable for TLCS.*

Given an instance of the reachability problem, defined by a TLCS $\mathcal{T} = \langle S, s_{init}, C, M, X, \Delta \rangle$ and a control state $s \in S$, we construct an equivalent instance of the reachability problem, defined by a DLCS $\mathcal{D} = \langle S^{\mathcal{D}}, s^{\mathcal{D}}_{init}, C^{\mathcal{D}}, \Sigma^{\mathcal{D}}, \Delta^{\mathcal{D}} \rangle$ (that we derive from $\mathcal{T}$) and the (same) control state s (as we shall see, all control states in S belong also to $S^{\mathcal{D}}$). The idea of the proof is inspired in parts by the region construction for timed automata [9]. A major difficulty in our case is the fact that we have unboundedly many ages to keep track of, and the fact that we also have to keep track of the ordering of an unbounded number of messages inside the channels. We will describe the ingredients of the encoding (the derivation of $\mathcal{D}$ from $\mathcal{T}$) step by step. First, we will introduce the set $C^{\mathcal{D}}$ of channels and the alphabet $\Sigma^{\mathcal{D}}$ for such channels, then we will define the encoding into a configuration of $\mathcal{D}$ of a configuration of $\mathcal{T}$. We will then define a set of meta-transitions, to aid us in the final task of this section, namely presenting how to simulate a run of $\mathcal{T}$ using our encoding $\mathcal{D}$.

Below, let k_{max} be the largest integer that occurs in the definition of any interval in Δ.

$\Sigma^{\mathcal{D}}$ and $C^{\mathcal{D}}$

As in the case of timed automata, we conclude that it is not meaningful to keep track of exact values of clocks and exact ages of messages beyond k_{max}. Each message in m with age r traveling inside a channel c in $\mathcal{T}$ will be encoded by a pair $\langle\langle c, m\rangle, j\rangle$ in $\mathcal{D}$ where $j = \mathsf{Int}(r)$ if $r \leq k_{max}$ and $j = \infty$ if $r > k_{max}$. The message m thus belongs to the set $\Sigma_m := (C \times M) \times \left([k_{max}]^0 \cup \{\infty\}\right)$. We will use three types of channels in $\mathcal{D}$ to store messages. First, we use a static channel c_0 to store messages whose ages are $\leq k_{max}$ and whose fractional parts are zero. Second, we use the dynamic part to store messages whose values are $\leq k_{max}$ and whose fractional parts are strictly positive. Messages stored in the same dynamic channels encode messages in $\mathcal{T}$ that have identical fractional parts. The fractional parts of messages inside different dynamic channels have increasing fractional parts as we move from left to right. Finally, we use a static channel c_∞ to store messages whose ages are $> k_{max}$.

We will also encode the clocks of $\mathcal{T}$ as messages in the channels of $\mathcal{D}$. To that end we define $\Sigma_x := X \times \left([k_{max}]^0 \cup \{\infty\}\right)$. A clock x will then be represented by a pair $\langle x, j\rangle$ that will be interpreted in a similar manner as above. Throughout the simulation, we will satisfy the invariant that at most one copy of each clock x will be present inside the channels of $\mathcal{D}$. For messages from the set $\Sigma_m \cup \Sigma_x$, we refer to the second component of the tuple as the *age* of the message.

Finally, for technical reasons, we will use a special *sentinel* message $\#$ and a *temporary channel* c_{tmp}. In summary we define $\Sigma^{\mathcal{D}} := \Sigma_m \cup \Sigma_x \cup \{\#\}$, and define $C^{\mathcal{D}} := \{c_0, c_\infty, c_{tmp}\}$.

Encoding of Configurations

We show how to abstract (encode) configurations of $\mathcal{T}$ by configurations of $\mathcal{D}$. For each configuration in $\mathcal{T}$ we will define a set $\alpha(\gamma)$ of configurations in $\mathcal{D}$. In our simulation, all these configurations will have equivalent behaviors and any one of them may be chosen to represent γ. The abstraction relies crucially on a property satisfied by all configurations that arise in a run of $\mathcal{T}$. More precisely, since $\mathcal{T}$ is normalized (cf. Section 3), the ages of messages inside any channel are sorted (if $\langle m_1, r_1\rangle$ is in on the left of $\langle m_2, r_2\rangle$ then $r_1 \leq r_2$). Furthermore, the ordering in which the messages occur inside the channel reflects the ordering in which they were sent to the channel (in particular, this holds even if $r_1 = r_2$).

We present the encoding in several steps. First, we define some operations on words $w \in \left(((C \times M) \cup X) \times \mathbb{R}^{\geq 0}\right)^*$. Let $r \in [0, 1)$ and $u = \langle \sigma'_1, a'_1\rangle \cdots \langle \sigma'_n, a'_n\rangle$ be the longest subword of w such that $\mathsf{Frac}(a'_i) = r$ for all i. We define the *fractional projection* of w with respect to r, written $w|_r$, as the word $\langle \sigma'_1, \mathsf{Int}(a'_1)\rangle \cdots \langle \sigma'_n, \mathsf{Int}(a'_n)\rangle$. In other words, $w|_r$ is obtained by (i) constructing the subword of w that consists of only pairs where the fractional part of the age is equal to r, and (ii) removing r from the age of each message in the sequence.

Consider a configuration $\gamma = \langle s, \nu, \omega\rangle$. We will partition the messages and the clocks depending on whether their ages exceed k_{max} or not. For a channel $c \in C$ such that $\nu(c) = (m_1, a_1)(m_2, a_2) \cdots (m_n, a_n)$, let k be the greatest i such that $a_i \leq k_{max}$. For ease of notation, we define the two words $c^{\leq k_{max}} := \langle\langle c, m_1\rangle, a_1\rangle \cdots \langle\langle c, m_k\rangle, a_k\rangle$ and $c^{>k_{max}} := \langle\langle c, m_{k+1}\rangle, \infty\rangle \cdots \langle\langle c, m_n\rangle, \infty\rangle$. Similarly we let $x^{\leq k_{max}} = \langle x_1, \mathtt{X}(x_1)\rangle \cdots \langle x_k, \mathtt{X}(x_k)\rangle$ where $x_1 \cdots x_k$ is an arbitrary enumeration of all $x \in X$ such that $\mathtt{X}(x) \leq k_{max}$. In the same manner, we define $x^{>k_{max}}$ as a word $\langle x_{k+1}, \infty\rangle \cdots \langle x_n, \infty\rangle$ where $x_{k+1} \cdots x_n$ is an arbitrary enumeration of all $x \in X$ such that $\mathtt{X}(x) > k_{max}$. Let $c_1, c_2, \ldots, c_l$ be an enumeration of C. We define $u := (c_1^{\leq k_{max}} \cdot c_2^{\leq k_{max}} \cdots c_l^{\leq k_{max}} \cdot x^{\leq k_{max}})$, i.e., u is the concatenation of the

parts of all the channels that has not exceeded k_{max}, and clocks that has not exceeded k_{max}. Finally, let $r_1 < r_2 \ldots < r_j$ be all strictly positive fractional parts occurring in some $c_i^{\leq k_{max}}$ or in $x^{\leq k_{max}}$.

Now we can define the abstraction of γ, written $\alpha(\gamma)$, as the set of all $d = \langle q, \nu, \omega \rangle$ where

- $q = s$
- ν is the function such that $\nu(c_0) = (u)|_0$, $\nu(c_\infty) = c_1^{>k_{max}} \cdot c_2^{>k_{max}} \cdots c_l^{>k_{max}} \cdot x^{>k_{max}}$ and $\nu(c_{tmp}) = \epsilon$.
- $\omega = \langle (u)|_{r_1} \rangle \cdots \langle (u)|_{r_j} \rangle$.

In other words: (i) the abstraction preserves the control state, (ii) all messages and clocks that are $\leq k_{max}$ and have zero fractional parts, are put in c_0, where the relative order of elements in the same channel is preserved, (iii) all messages and clocks that are $> k_{max}$ are put in c_∞, again with relative order preserved, and (iv) the dynamic channel vector is constructed by building a word for each positive fractional part, and order them by these fractional parts.

Intuitively, the abstraction preserves the following invariants:

- Any message or clock with an age not greater than k_{max} is translated into a message consisting of the same message or clock, and and its original age with the fractional part stripped.
- Any two messages, a message and a clock, or two clocks, with age less than or equal to k_{max} will end up in the same channel in the abstracted system if and only if they have the same fractional part of their age in $\mathcal{T}$. For pairs of messages from the same channel in $\mathcal{T}$, their relative order in the channel in $\mathcal{D}$ will be the same as in $\mathcal{T}$.
- For any two messages, a message and a clock, or two clocks, with age less than or equal to k_{max}, the one with the greater fractional part will end up to the right of one with the smaller fractional part.
- Any two messages with an age greater than k_{max} will end up in the c_∞, with their relative order preserved.

Meta-Transitions

We start by defining some meta-transitions for the DLCS, allowing us to more compactly describe the simulation. Due to space restrictions, we only provide an overview of the construction here, for more details see [2]. Each meta-transition consists of a finite set of ordinary DLCS transitions, possibly containing loops and passing through a number of temporary states. Note that even though the meta transitions might cause an execution of our system to block because of picking the wrong branch in some nondeterministic choice, this is not a problem since we are only interested in the study of safety properties. The meta-transitions are defined as follows:

- $\mathsf{empty}(c)$: empties the channel c, by receiving all possible messages.
- $\mathsf{copy}(c_1, c_2)$: copies the content of channel c_1 into channel c_2, overwriting any previous content, while c_1 remains unchanged.
- $\mathsf{filter}(c, \Sigma)$: filters the channel c, such that only elements from Σ remain.
- $\mathsf{map}(c, f)$, acts on the channel c by replacing each message σ with $f(\sigma)$.
- $\mathsf{HasElementsFrom}(c, \Sigma)$: enforces that there is at least one element in the channel c from the set Σ. If this is not the case, the simulation blocks. $\mathsf{HasElementsFrom}(\omega, \Sigma)$ performs the same operation on the set of dynamic channels rather than on a static channel c.

- $\mathsf{HasNoElementsFrom}(c, \Sigma)$, enforces that there no element in the channel c from the set Σ. If this is not the case, the simulation blocks. $\mathsf{HasNoElementsFrom}(\omega, \Sigma)$ is defined analogously.
- $\mathsf{ReceiveFromSet}(c, m, \Sigma)$ receives (deletes) the message m from c but only if the following condition holds. Search for the first (rightmost) occurrence of a message $m' \in \Sigma$ in c. If $m' = m$ then it is deleted. If $m' \neq m$ or c does not contain any messages from Σ, the simulation blocks. $\mathsf{ReceiveFromSet}(\omega, m, \Sigma)$ is defined analogously for the dynamic part, namely the search is carried out through all the channels from right to left. For a given channel, we search from right to left.

Simulation of Discrete Transitions

Each transition $t = \langle s_1, op, s_2 \rangle \in \Delta$ is simulated using a set of transitions in $\Delta^{\mathcal{D}}$ as follows:

- If $t = \mathsf{nop}$, we let $\langle s_1, \mathsf{nop}, s_2 \rangle \in \Delta^{\mathcal{D}}$.
- If $t = c!m$, we let $\langle s_1, c_0!\langle\langle c, m\rangle, 0\rangle, s_2\rangle \in \Delta^{\mathcal{D}}$. In other words, we send the message m, tagged with the identity of the channel, to c_0. This reflects the fact that initial ages of messages are set to 0 (since $\mathcal{T}$ is normalized).
- If $t = c?m \in I$. This is the most complicated case. We need to search the dynamic channels and also the static channels c_0 and c_∞ in $\mathcal{D}$ in order to find the message corresponding to the rightmost message in c. If this message is m then we delete it, otherwise we block the simulation. This is carried out in two steps, namely (i) *guessing*: we non-deterministically "guess" the age of the message, and (ii) *checking*: for the given guess, we check that there are no other messages in channel c that are older than the current one. Concretely, in the *guessing* step we assume that the message has an age which is either (i) $k \in [k_{max}]^0$ for some integer $k \in I$, or (ii) in the interval $(k, k+1)$ for some $k \in [k_{max} - 1]^0$ where $(k, k+1) \subseteq I$, or (iii) in the interval (k_{max}, ∞) if $(k_{max}, \infty) \subseteq I$. Let $\Sigma_1 = (\{c\} \times M) \times \{\infty\})$, $\Sigma_2 = ((\{c\} \times M) \times \{\ell \mid k \leq \ell \leq k_{max}\})$, $\Sigma_3 = ((\{c\} \times M) \times \{\ell \mid k < \ell \leq k_{max}\})$ and $\Sigma_4 = ((\{c\} \times M) \times \{k\})$. The *checking* step is carried out depending on the guessed age of the message as follows.
  - Guess $k \in [k_{max}]^0$. We use the operations (i) $\mathsf{HasNoElementsFrom}(c_\infty, \Sigma_1)$, (ii) $\mathsf{HasNoElementsFrom}(\omega, \Sigma_2)$, and (iii) $\mathsf{HasNoElementsFrom}(c_0, \Sigma_3)$, to ensure that c does not contain any message older than m. Then, use $\mathsf{ReceiveFromSet}(c_0, m, \Sigma_4)$ to try to receive m.
  - Guess $(k, k+1)$ for some $k \in [k_{max} - 1]^0$. We use (i) $\mathsf{HasNoElementsFrom}(c_\infty, \Sigma_1)$, (ii) $\mathsf{HasNoElementsFrom}(\omega, \Sigma_3)$, and (iii) $\mathsf{HasNoElementsFrom}(c_0, \Sigma_3)$ to ensure that c does not contain any message older than m. Then, use $\mathsf{ReceiveFromSet}(\omega, m, \Sigma_4)$ to try to receive m.
  - Guess (k_{max}, ∞). Use $\mathsf{ReceiveFromSet}(c_\infty, m, \Sigma_1)$ to try to receive m.
- If $t = x \in I$ then we guess the value of x according to one of the three forms described in the previous case. Since we satisfy the invariant that there is at most one message representing x in the channels of $\mathcal{D}$, the simulation is simpler in this case. More precisely, if we guess the age of x to be k for some $k \in [k_{max}]^0$ then we use $\mathsf{HasElementsFrom}(c_0, \{\langle x, k\rangle\})$. If we guess $(k, k+1)$ for some $k \in [k_{max} - 1]^0$ then we use $\mathsf{HasElementsFrom}(\omega, \{\langle x, k\rangle\})$. Finally, if we guess (k_{max}, ∞) then we use $\mathsf{HasElementsFrom}(c_\infty, \{\langle x, \infty\rangle\})$.
- If $t = x \leftarrow 0$, we simply remove the message representing x from the channels of $\mathcal{D}$, and then send it again with age 0 to c_0. Concretely, we non-deterministically use $\mathsf{ReceiveFromSet}(c_0, \langle x, i\rangle, (\{x\} \times [k_{max}]^0))$, $\mathsf{ReceiveFromSet}(\omega, (x, i), (\{x\} \times [k_{max}]^0))$, or $\mathsf{ReceiveFromSet}(c_\infty, \langle x, \infty\rangle, \{\langle x, \infty\rangle\})$ where $i \in [k_{max}]^0$. After that, we know that we

have no message representing x in the channels of $\mathcal{D}$ anymore, so we add an operation $c_0!\langle x, 0\rangle$ to send $\langle x, 0\rangle$ to c_0. The clock has now been reset.

Simulating Timed Transitions

We show how to simulate timed transitions of the form $\langle s, \mathtt{X}, \nu\rangle \xrightarrow{\delta}_{\mathcal{T}} \langle s, \mathtt{X}', \nu'\rangle$ for some $\delta > 0$. We distinguish between two cases, namely (i) there is at least one message or clock with value ($\leq k_{max}$) and a zero fractional part (i.e., $c_0 \neq \epsilon$), and (ii) that no such message or clock exists (i.e., $c_0 = \epsilon$):

- In the first case, we can let time pass by a sufficiently small real number, such that no clock with a positive fractional part before the transition reaches the next integer value after the transition. The contents of c_0 will be divided between messages that will be transferred to c_∞ (representing message ages and clocks values equal to k_{max}); and messages that will be placed in a new channel at the leftmost position in the dynamic part (representing message ages and clock values $< k_{max}$). Concretely, we perform the following steps: (i) we use $\mathsf{copy}(c_0, c_{tmp})$ to copy the contents of c_0 to the temporary channel c_{tmp}. (ii) we use $\mathsf{filter}(c_{tmp}, \Sigma_1)$ where $\Sigma_1 = (X \times \{k_{max}\}) \cup ((C \times M) \times \{k_{max}\})$ to only keep messages with ages equal to k_{max} in c_{tmp}. (iii) We send the messages of c_{tmp} one after one to c_∞, changing the second component from k_{max} to ∞ for each message. (iv) We use $\mathsf{filter}(c_0, \Sigma_2)$ where $\Sigma_2 = \left(X \times [k_{max} - 1]^0\right) \cup \left((C \times M) \times [k_{max} - 1]^0\right)$ to only keep messages with ages $< k_{max}$ in c_0. (v) We send the content of c_0 to the dynamic part using $send\_channel(c_0)$. (vi) We use $\mathsf{empty}(c_0)$ to empty c_0.
- In the second case, we let time pass by exactly the amount needed to make the clock values and the message ages in the rightmost dynamic channel equal to the next integer. Let $f \in ((\Sigma_m \cup \Sigma_x) \to (\Sigma_m \cup \Sigma_x))$ be a function such that $f(\langle\langle c, m\rangle, i\rangle) = \langle\langle c, m\rangle, i + 1\rangle$ and $f(\langle x, i\rangle) = \langle x, i + 1\rangle$ for any $c \in C$, $m \in M$, $x \in X$, and $i \in [k_{max} - 1]^0$. We use $receive\_channel(c_0)$ to move the contents of the rightmost dynamic channel to c_0. Then, we use $\mathsf{map}(c_0, f)$ to increase the integer parts of ages of clocks and messages by one.

Simulating Lossy Transitions

Since we have lossiness in $\mathcal{D}$, the simulation is immediate.

6 Conclusions, Discussion, and Future Work

We have shown the decidability of the reachability problem for TLCS, a model that extends both lossy channel systems and timed automata. To this end, we have introduced a new model, namely DLCS that operates on second-order lossy channels. We believe that DLCS are interesting in their own right. In fact, we can define higher-order LCS that contain "nested channels of channels" of arbitrary depth, in a similar manner to higher-order pushdown automata [20]. It is straightforward to extend the method we present in this paper to show that transition systems induced by higher-order LCS are also well quasi-ordered and hence their reachability problem is decidable. To simplify the presentation (and since it suffices for our purposes) we have chosen to present the proof only for the case where the hierarchy is restricted to two levels (i.e., DLCS).

The proof techniques we provide in this paper are entirely different from the ones earlier presented for other timed models. For instance, decidability of the reachability (coverability) problem for timed Petri nets [5] is achieved by directly proving that the induced transition system is well quasi-ordered. In particular, in contrast to our method, the proof does not

rely on a translation to an untimed model. On the other hand, the proof for timed pushdown systems [3] reduces the problem to the underlying untimed model, i.e., (untimed) pushdown automata. Although, we here provide a reduction to an untimed model, the target model is more powerful than the original one (DLCS vs. plain LCS). Indeed, we believe that a translation from TLCS to plain LCS that preserves reachability properties is not possible.

As future work, we will consider probabilistic and game extensions of the current model.

References

1. P. A. Abdulla. Well (and better) quasi-ordered transition systems. *The Bulletin of Symbolic Logic*, 16(4):457–515, 2010.
2. P. A. Abdulla, M. F. Atig, and J. Cederberg. Timed lossy channel systems. Technical Report 2012-031, Department of Information Technology, Uppsala University, October 2012.
3. P. A. Abdulla, M. F. Atig, and J. Stenman. Dense-timed pushdown automata. In *LICS*. IEEE Computer Society, 2012.
4. P. A. Abdulla, M. F. Atig, and J. Stenman. The minimal cost reachability problem in priced timed pushdown systems. In *LATA*, volume 7183 of *LNCS*, 2012.
5. P. A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *ICATPN*, 2001.
6. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Carl Leonardsson, and Ahmed Rezine. Counter-example guided fence insertion under tso. In *TACAS*, 2012.
7. Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
8. Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *LICS*, pages 160–170. IEEE Computer Society, 1993.
9. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
10. M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, pages 7–18. ACM, 2010.
11. M. Benerecetti, S. Minopoli, and A. Peron. Analysis of timed büchi state machines. In *TIME*, pages 61–68. IEEE Computer Society, 2010.
12. B. Bérard, F. Cassez, S. Haddad, O. Roux, and D. Lime. Comparison of different semantics for time Petri nets. In *ATVA 2005*, 2005.
13. A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems*, LNCS 999, pages 64–85. Springer, 1994.
14. P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *FSTTCS*, LNCS 3328, pages 148–160. Springer, 2004.
15. P. Bouyer and F. Laroussinie. Model checking timed automata. In Stephan Merz and Nicolas Navet, editors, *Modeling and Verification of Real-Time Systems*, pages 111–140. ISTE Ltd. – John Wiley & Sons, Ltd., January 2008.
16. Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *LICS*, pages 205–216. IEEE Computer Society Press, 2008.
17. Z. Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theor. Comput. Sci.*, 302(1-3):93–121, 2003.
18. Z. Dang, T. Bultan, O. H. Ibarra, and R. A. Kemmerer. Past pushdown timed automata and safety verification. *Theor. Comput. Sci.*, 313(1):57–71, 2004.
19. M. Emmi and R. Majumdar. Decision problems for the verification of real-time software. In *HSCC*, LNCS 3927, pages 200–211. Springer, 2006.
20. M. Hague and L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. *Logical Methods in Computer Science*, 4(4), 2008.

21 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7):326–336, 1952.
22 A. Trivedi and D. Wojtczak. Recursive timed automata. In *ATVA*, pages 306–324, 2010.

Solving the Canonical Representation and Star System Problems for Proper Circular-Arc Graphs in Logspace

Johannes Köbler, Sebastian Kuhnert*, and Oleg Verbitsky†

Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
{koebler,kuhnert,verbitsk}@informatik.hu-berlin.de

Abstract

We present a logspace algorithm that constructs a canonical intersection model for a given proper circular-arc graph, where *canonical* means that isomorphic graphs receive identical models. This implies that the recognition and the isomorphism problems for these graphs are solvable in logspace. For the broader class of concave-round graphs, which still possess (not necessarily proper) circular-arc models, we show that a canonical circular-arc model can also be constructed in logspace. As a building block for these results, we design a logspace algorithm for computing canonical circular-arc models of circular-arc hypergraphs; this important class of hypergraphs corresponds to matrices with the *circular ones property*.

Furthermore, we consider the Star System Problem that consists in reconstructing a graph from its closed neighborhood hypergraph. We show that this problem is solvable in logarithmic space for the classes of proper circular-arc, concave-round, and co-convex graphs.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Proper circular-arc graphs, graph isomorphism, canonization, circular ones property, logspace complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.387

1 Introduction

With a family of sets $\mathcal{H}$ we associate the *intersection graph* $\mathbb{I}(\mathcal{H})$ on vertex set $\mathcal{H}$ where two sets $A, B \in \mathcal{H}$ are adjacent if and only if they have a non-empty intersection. We call $\mathcal{H}$ an *intersection model* of a graph G if G is isomorphic to $\mathbb{I}(\mathcal{H})$. Any isomorphism from G to $\mathbb{I}(\mathcal{H})$ is called a *representation* of G by an intersection model. If $\mathcal{H}$ consists of intervals (resp. arcs of a circle), it is also referred to as an *interval model* (resp. an *arc model*). An intersection model $\mathcal{H}$ is *proper* if the sets in $\mathcal{H}$ are pairwise incomparable by inclusion. G is called a *(proper) interval graph* if it has a (proper) interval model. The classes of *circular-arc* and *proper circular-arc* graphs are defined similarly. Throughout the paper we will use the shorthands *CA* and *PCA*, respectively.

We design a logspace algorithm that for a given PCA graph computes a canonical representation by a proper arc model, where *canonical* means that isomorphic graphs receive

* Supported by DFG grant KO 1053/7–1.
† Supported by DFG grant VE 652/1–1. This work was initiated under support by the Alexander von Humboldt Fellowship. On leave from the Institute for Applied Problems of Mechanics and Mathematics, Lviv, Ukraine.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 387–399

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

identical models. Note that this algorithm provides a simultaneous solution in logspace of both the recognition and the isomorphism problems for the class of PCA graphs.

In [18], along with Bastian Laubner we gave a logspace solution for the canonical representation problem of proper interval graphs. Though PCA graphs may at first glance appear close relatives of proper interval graphs, the extension of the result of [18] achieved here is far from being straightforward. Differences between the two classes of graphs are well known and have led to different algorithmic approaches also in the past; e.g. [11, 17, 22]. One difference, very important in our context, lies in the relationship of these graph classes to interval and circular-arc hypergraphs that we will explain shortly.

An *interval hypergraph* is a hypergraph isomorphic to a system of intervals of integers. A *circular-arc (CA) hypergraph* is defined similarly if, instead of integer intervals, we consider arcs in a discrete cycle. With any graph G, we associate its *closed neighborhood hypergraph* $\mathcal{N}[G] = \{N[v]\}_{v \in V(G)}$ on the vertex set of G, where for each vertex v we have the hyperedge $N[v]$ consisting of v and all vertices adjacent to v. Roberts [27] discovered that G is a proper interval graph if and only if $\mathcal{N}[G]$ is an interval hypergraph. The circular-arc world is more complex. While $\mathcal{N}[G]$ is a CA hypergraph whenever G is a PCA graph, the converse is not always true. PCA graphs are properly contained in the class of those graphs whose neighborhood hypergraphs are CA. Graphs with this property are called *concave-round* by Bang-Jensen, Huang, and Yeo [3] and *Tucker graphs* by Chen [8]. The latter name is justified by Tucker's result [29] saying that all these graphs are CA (although not necessarily proper CA). Hence, it is natural to consider the problem of constructing arc representations for concave-round graphs. We solve this problem in logspace and also in a canonical way.

Our working tool is a logspace algorithm for computing a canonical representation of CA hypergraphs. This algorithm can also be used to test in logspace whether a given Boolean matrix has the *circular ones property*, that is, whether the columns can be permuted so that the 1-entries in each row form a segment up to a cyclic shift. Note that a matrix has this property if and only if it is the incidence matrix of a CA hypergraph. The recognition problem of the circular ones property arises in computational biology, namely in analysis of circular genomes [13, 25].

Our techniques are also applicable to the *Star System Problem* where, for a given hypergraph $\mathcal{H}$, we have to find a graph G such that $\mathcal{H} = \mathcal{N}[G]$, if such a graph exists. In the restriction of the problem to a class of graphs C, we seek for G only in C. We give logspace algorithms solving the Star System Problem for PCA and for concave-round graphs.

Comparison with previous work. The recognition problem for PCA graphs, along with model construction, was solved in linear time by Deng, Hell, and Huang [11] and by Kaplan and Nussbaum [17]; and in AC^2 by Chen [7]. Note that linear-time and logspace results are in general incomparable, while the existence of a logspace algorithm for a problem implies that it is solvable in AC^1. The isomorphism problem for PCA graphs was solved in linear time by Lin, Soulignac, and Szwarcfiter [22]. In a very recent paper [10], Curtis et al. extend this result to concave-round graphs.

The isomorphism problem for concave-round graphs was solved in AC^2 by Chen [8]. Circular-arc models for concave-round graphs were known to be constructible also in AC^2 (Chen [6]).

Extending these complexity upper bounds to the class of all CA graphs remains a challenging problem. While this class can be recognized in linear time by McConnell's algorithm [24] (along with constructing an intersection model), no polynomial-time isomorphism test for CA graphs is currently known (see the discussion in [10], where a counterexample to the

correctness of Hsu's algorithm [14] is given). This provides further evidence that CA graphs are algorithmically harder than interval graphs. For the latter class we have linear-time algorithms for both recognition and isomorphism due to the seminal work by Booth and Lueker [4, 23], and a canonical representation algorithm taking logarithmic space is designed in [18].

The aforementioned circular ones property and the related *consecutive ones property* were studied in [4, 15, 16], where linear-time algorithms are given; parallel AC^2 algorithms were suggested in [9, 2].

The decision version of the Star System Problem is in general NP-complete (Lalonde [21]). It stays NP-complete if restricted to non-co-bipartite graphs (Aigner and Triesch [1]) or to H-free graphs for H being a cycle or a path on at least 5 vertices (Fomin et al. [12]). The restriction to co-bipartite graphs has the same complexity as the general graph isomorphism problem [1]. Polynomial-time algorithms are known for H-free graphs for H being a cycle or a path on at most 4 vertices [12] and for bipartite graphs (Boros et al. [5]). An analysis of the algorithms in [12] for C_3- and C_4-free graphs shows that the Star System Problem for these classes is solvable even in logspace, and the same holds true for the class of bipartite graphs; see [20]. Moreover, the problem is solvable in logspace for any logspace-recognizable class of C_4-free graphs, in particular, for chordal, interval, and proper interval graphs; see [20].

Due to space limitations some proofs are only sketched; full details can be found in an e-print [19].

2 Basic definitions

The vertex set of a graph G is denoted by $V(G)$. The *complement of a graph* G is the graph $\overline{G}$ with $V(\overline{G}) = V(G)$ such that two vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G. The set of all vertices at distance at most (resp. exactly) 1 from a vertex $v \in V(G)$ is called the *closed* (resp. *open*) *neighborhood* of v and denoted by $N[v]$ (resp. $N(v)$). Note that $N[v] = N(v) \cup \{v\}$. We call vertices u and v *twins* if $N[u] = N[v]$ and *fraternal vertices* if $N(u) = N(v)$. A vertex u is *universal* if $N[u] = V(G)$.

The *canonical labeling problem* for a class of graphs C is, given a graph $G \in \mathsf{C}$ with n vertices, to compute a map $\lambda_G \colon V(G) \to \{1, \ldots, n\}$ so that the graph $\lambda_G(G)$, the image of G under λ_G on the vertex set $\{1, \ldots, n\}$, is the same for isomorphic input graphs. We say that λ_G is a *canonical labeling* and that $\lambda_G(G)$ is a *canonical form* of G.

Recall that a *hypergraph* is a pair $(X, \mathcal{H})$, where X is a set of vertices and $\mathcal{H}$ is a family of subsets of X, called *hyperedges*. We will use the same notation $\mathcal{H}$ to denote a hypergraph and its hyperedge set and, similarly to graphs, we will write $V(\mathcal{H})$ referring to the vertex set X of the hypergraph $\mathcal{H}$. We will allow *multiple hyperedges*; in this case an isomorphism has to respect multiplicities.

The *complement of a hypergraph* $\mathcal{H}$ is the hypergraph $\overline{\mathcal{H}} = \{\overline{H}\}_{H \in \mathcal{H}}$ on the same vertex set, where $\overline{H} = V(\mathcal{H}) \setminus H$. Each hyperedge $\overline{H}$ of $\overline{\mathcal{H}}$ inherits the multiplicity of H in $\mathcal{H}$. We associate with a graph G two hypergraphs defined on the vertex set $V(G)$. The *closed* (resp. *open*) *neighborhood hypergraph* of G is defined by $\mathcal{N}[G] = \{N[v]\}_{v \in V(G)}$ (resp. by $\mathcal{N}(G) = \{N(v)\}_{v \in V(G)}$). *Twins in a hypergraph* are two vertices such that every hyperedge contains either both or none of them. Note that two vertices are twins in $\mathcal{N}[G]$ if and only if they are twins in G.

By $\mathbb{C}_n$ we denote the directed cycle on the vertex set $\{1, \ldots, n\}$ with arrows from i to $i+1$ and from n to 1. An *arc* A is either empty ($A = \emptyset$), complete ($A = \{1, \ldots, n\}$), or a segment $A = [a^-, a^+]$ with *extreme points* a^- and a^+ that consists of the points appearing in the

directed path from a^- to a^+ in the cycle $\mathbb{C}_n$. An *arc system* $\mathcal{A}$ is a hypergraph whose vertex set is $\{1,\dots,n\}$ and whose hyperedges are arcs of $\mathbb{C}_n$. $\mathcal{A}$ is *tight* if any two arcs $A=[a^-,a^+]$ and $B=[b^-,b^+]$ in $\mathcal{A}$ have the following property: if $\emptyset \neq A \subseteq B \neq \mathbb{C}_n$, then $a^- = b^-$ or $a^+ = b^+$.

An *arc representation* of a hypergraph $\mathcal{H}$ is an isomorphism ρ from $\mathcal{H}$ to an arc system $\mathcal{A}$. The arc system $\mathcal{A}$ is referred to as an *arc model* of $\mathcal{H}$. The notions of an *interval representation* and an *interval model* of a hypergraph are introduced similarly, where *interval* means an interval of integers. Hypergraphs having arc representations are called *circular-arc (CA) hypergraphs*, and those having interval representations are called *interval hypergraphs*.

We call a CA hypergraph *tight*, if it admits a tight arc model. Recognition of tight CA hypergraphs reduces to recognition of CA hypergraphs. To see this, given a hypergraph $\mathcal{H}$, define its *tightened hypergraph* $\mathcal{H}^{\Subset}$ by $\mathcal{H}^{\Subset} = \mathcal{H} \cup \{ A \setminus B : A,B \in \mathcal{H}\}$. Then $\mathcal{H}$ is a tight CA hypergraph if and only if $\mathcal{H}^{\Subset}$ is a CA hypergraph (for if $A,B \in \mathcal{H}$ and $\emptyset \neq B \subsetneq A$, then B cannot be an inner part of A in any arc model of $\mathcal{H}^{\Subset}$).

A *circular order* on a finite set $S=\{s_1,\dots,s_n\}$ is described by a circular successor relation $\prec$ on S (meaning that the digraph $(S,\prec)$ is a cycle). Each arc representation ρ of a CA hypergraph $\mathcal{H}$ induces a circular order $\prec$ on $V(\mathcal{H})$ such that the hyperedges in $\mathcal{H}$ are arcs w.r.t. $\prec$; a circular order on $V(\mathcal{H})$ with this property is called a *CA order* of $\mathcal{H}$. Conversely, each CA order of $\mathcal{H}$ specifies an arc representation of $\mathcal{H}$ up to rotation. Where rotations are not important, we will describe arc representations by CA orders. We call a CA order of $\mathcal{H}$ *tight*, if it makes $\mathcal{H}$ a tight arc system. Similarly, we will use the notion of *(tight) interval orders* in the case of interval hypergraphs.

Given a CA order $\prec$ of a hypergraph $\mathcal{H}$, consider the set of all arcs $A \subset V(\mathcal{H})$ w.r.t. $\prec$ excepting the empty arc $\emptyset$ and the complete arc $V(\mathcal{H})$. The CA order $\prec$ induces a (lexicographic) circular order $\prec^*$ on this set, where $A \prec^* B$ if $a^- = b^-$ and $a^+ \prec b^+$ or if $a^- \prec b^-$, $|A| = n-1$, and $|B|=1$. By "restricting" $\prec^*$ to the set $\mathcal{H}$ (assuming that $\emptyset, V(\mathcal{H}) \notin \mathcal{H}$) we obtain a circular order $\prec_{\mathcal{H}}$ on $\mathcal{H}$: For $A,B \in \mathcal{H}$ we define $A \prec_{\mathcal{H}} B$ if either $A \prec^* B$ or there exist arcs $X_1,\dots,X_k \notin \mathcal{H}$ such that $A \prec^* X_1 \prec^* \dots \prec^* X_k \prec^* B$. We say that the circular order $\prec_{\mathcal{H}}$ on $\mathcal{H}$ is *lifted from* the CA order $\prec$ on $V(\mathcal{H})$.

An *arc representation of a graph* G is an isomorphism $\alpha\colon V(G) \to \mathcal{A}$ from G to the intersection graph $\mathbb{I}(\mathcal{A})$ of an arbitrary arc system $\mathcal{A}$. If $\emptyset, V(\mathcal{A}) \notin \mathcal{A}$ (this always holds when G has neither an isolated nor a universal vertex), we use the lifted circular order $\prec_{\mathcal{A}}$ on $\mathcal{A}$ to define a circular order $\prec_\alpha$ on $V(G)$, where $u \prec_\alpha v$ if and only if $\alpha(v) \prec_{\mathcal{A}} \alpha(u)$. We call $\prec_\alpha$ the *geometric order* on $V(G)$ associated with α.

Roadmap. In Section 3 we desribe a logspace algorithm for computing a canonical arc representation of a given CA hypergraph. In Section 4 we show that for non-co-bipartite PCA graphs G, the neighborhood hypergraph $\mathcal{N}[G]$ admits a unique CA order, which coincides with the geometric order $\prec_\alpha$ for any proper arc representation α of G. In Section 5, we employ these facts to compute canonical representations of non-co-bipartite PCA graphs in logspace. To achieve the same for co-bipartite PCA graphs G (and all concave-round graphs), we use the fact that $\mathcal{N}(\overline{G})$ is in this case an interval hypergraph and show how to convert an interval representation of $\mathcal{N}(\overline{G})$ into an arc representation of G. Finally, in Section 6 we apply the techniques of Sections 3 and 4 to the Star System Problem.

3 Canonical arc representations of hypergraphs

▶ **Theorem 1.** *The canonical representation problem for CA hypergraphs is solvable in logspace.*

Proof sketch. We prove this result by a logspace reduction to the canonical representation problem for edge-colored interval hypergraphs, which is already known to be in logspace [18]. Let $\mathcal{H}$ be an input CA hypergraph with n vertices. For each vertex $x \in V(\mathcal{H})$ we construct the hypergraph $\mathcal{H}_x = \{H_x\}_{H\in\mathcal{H}}$ on the same vertex set, where $H_x = H$ if $x \notin H$ and $H_x = \overline{H}$ otherwise. Observe that every $\mathcal{H}_x$ is an interval hypergraph; cf. [29, Theorem 1]. Canonizing each $\mathcal{H}_x$ using the algorithm from [18], we obtain n interval representations $\rho_x\colon V(\mathcal{H}) \to \{1,\dots,n\}$; recall that $V(\mathcal{H}_x) = V(\mathcal{H})$. Each ρ_x gives us an arc model $\rho_x(\mathcal{H})$ of $\mathcal{H}$, which is obtained from the corresponding canonical interval model $\rho_x(\mathcal{H}_x)$ of $\mathcal{H}_x$ by complementing the intervals corresponding to complemented hyperedges. Among these n candidates, we choose the lexicographically least arc model as canonical and output the corresponding arc representation ρ_x.

There is a subtle point in this procedure: We need to distinguish between complemented and non-complemented hyperedges when canonizing $\mathcal{H}_x$; otherwise reversing the complementation could lead to non-equal models for isomorphic CA hypergraphs. For this reason we endow each interval hypergraph $\mathcal{H}_x$ with the edge-coloring $c_x\colon \mathcal{H}_x \to \{0,1\}$, where $c_x(H_x) = 1$ if $x \in H$ and $c_x(H_x) = 0$ otherwise. If both H and $\overline{H}$ are present in $\mathcal{H}$, this results in "multi-hyperedges" that have different colors. ◀

The *canonical labeling problem* for a class of hypergraphs C is defined exactly as for graphs. The canonical representation algorithm given by Theorem 1 also solves the canonical labeling problem for CA hypergraphs in logarithmic space. We conclude this section with noting that it can also be used to compute a canonical labeling for the duals of CA hypergraphs; this will be needed in Section 6.

Given a hypergraph $\mathcal{H}$ and a vertex $v \in V(\mathcal{H})$, let $v^* = \{\, H \in \mathcal{H} : v \in H\}$. The hypergraph $\mathcal{H}^* = \{\, v^* : v \in V(\mathcal{H})\}$ on the vertex set $V(\mathcal{H}^*) = \mathcal{H}$ is called the *dual hypergraph* of $\mathcal{H}$ (multiple hyperedges in $\mathcal{H}$ become twin vertices in $\mathcal{H}^*$). The map $\varphi\colon v \mapsto v^*$ is an isomorphism from $\mathcal{H}$ to $(\mathcal{H}^*)^*$. If $\mathcal{H}^*$ is a CA hypergraph, this map can be combined with a canonical labeling λ of $\mathcal{H}^*$ in order to obtain a canonical labeling $\hat{\lambda}$ of $\mathcal{H}$. More precisely, $\hat{\lambda}$ is obtained from the map $\lambda'(v) = \{\, \lambda(H) : v \in H\}$ by sorting and renaming the values of λ'.

▶ **Corollary 2.** *The canonical labeling problem for hypergraphs whose duals are CA can be solved in logspace.*

4 Linking PCA graphs and tight CA hypergraphs

In this section, we establish the connections between some classes of CA graphs and CA hypergraphs that will be used in the design of our algorithms.

Bang-Jensen et al. [3] call a graph G *concave-round* (resp. *convex-round*) if $\mathcal{N}[G]$ (resp. $\mathcal{N}(G)$) is a CA hypergraph. Since $\overline{\mathcal{N}[G]} = \mathcal{N}(\overline{G})$, concave-round and convex-round graphs are co-classes. Using this terminology, a result of Tucker [29] says that PCA graphs are concave-round, and concave-round graphs are CA.

To connect the canonical representation problem for PCA and concave-round graphs to that of CA hypergraphs, we use the fact that the graph classes under consideration can be characterized in terms of neighborhood hypergraphs. For concave-round graphs, this

directly follows from their definition, and we can find accompanying hypergraphs also for PCA graphs.

▶ **Theorem 3.** *A graph G is PCA if and only if $\mathcal{N}[G]$ is a tight CA hypergraph.*

The forward direction of Theorem 3 follows from Lemma 4 below. To prove the other direction, we distinguish two cases. If $\overline{G}$ is bipartite, then any tight arc model for $\mathcal{N}[G]$ can be transformed into a proper arc model for G, as described in Section 5. If $\overline{G}$ is not bipartite, then a result of Tucker [29] says that G is a PCA graph whenever $\mathcal{N}[G]$ is a CA hypergraph.

▶ **Lemma 4.** *The geometric order $\prec_\alpha$ on $V(G)$ associated with a proper arc representation α of a PCA graph G is a tight CA order for the hypergraph $\mathcal{N}[G]$.*

Proof. Let G be a PCA graph and let $\alpha\colon V(G) \to \mathcal{A}$ be a proper arc representation of G. We first show that the neighborhood $N[u]$ of any vertex $u \in V(G)$ is an arc w.r.t. to the order $\prec_\alpha$. If u is universal, the claim is trivial. Otherwise, let $\alpha(u) = [a^-, a^+]$. We split $N(u)$ in two parts, namely $N^-(u) = \{\, v \in N(u) : a^- \in \alpha(v)\}$ and $N^+(u) = \{\, v \in N(u) : a^+ \in \alpha(v)\}$. Indeed, no vertex v is contained in both $N^-(u)$ and $N^+(u)$. Otherwise, since $\mathcal{A}$ is proper, the arcs $\alpha(v)$ and $\alpha(u)$ would cover the whole cycle, both intersecting any other arc $\alpha(w)$, contradicting the assumption that u is non-universal.

Now let $v \in N^+(u)$ and assume that $u \prec_\alpha v_1 \prec_\alpha \ldots \prec_\alpha v_k \prec_\alpha v$. We claim that every vertex v_i is in $N^+(u)$. Indeed, by the definition of $\prec_\alpha$, we have $\alpha(u) \prec_\mathcal{A} \alpha(v_1) \prec_\mathcal{A} \ldots \prec_\mathcal{A} \alpha(v_k) \prec_\mathcal{A} \alpha(v)$. If $\alpha(v) = [c^-, c^+]$ and $\alpha(v_i) = [b^-, b^+]$, we see that $b^- \in (a^-, c^-)$, $b^+ \in (a^+, c^+)$ and, hence, $a^+ \in [b^-, b^+]$. It follows that $N^+(u) \cup \{u\}$ is an arc starting at u. By a symmetric argument, $N^-(u) \cup \{u\}$ is an arc ending at u. Hence, also $N[u]$ is an arc, implying that $\prec_\alpha$ is a CA order for $\mathcal{N}[G]$.

It remains to show that the CA order $\prec_\alpha$ is tight. Suppose that $N[u] = [u^-, u^+] \subseteq N[v] = [v^-, v^+]$ and v is non-universal with $\alpha(v) = [c^-, c^+]$. Let's first assume that $u \in N^+(v) = (v, v^+]$. Since $u, v^+ \in N^+(v)$, it follows that $c^+ \in \alpha(u) \cap \alpha(v^+)$. Hence, u and v^+ are adjacent or equal, which implies that $u^+ = v^+$. If $u \in [v^-, v)$, a symmetric argument shows that $u^- = v^-$. ◀

Theorem 3 suggests that, given a tight CA order of $\mathcal{N}[G]$, we can use it to construct a proper arc model for G. For this we need the converse of Lemma 4. In the case that $\overline{G}$ is not bipartite, the following lemma implies that indeed each CA order of $\mathcal{N}[G]$ is the geometric order of some proper arc representation of G.

▶ **Lemma 5.** *If G is a connected twin-free PCA graph and $\overline{G}$ is not bipartite, then $\mathcal{N}[G]$ has a unique CA order up to reversing.*

The proof of Lemma 5 is based on a result of Deng, Hell, and Huang [11, Corollary 2.9]. We need some special properties of CA orders $\prec$ of $\mathcal{N}[G]$ when G is a PCA graph with non-bipartite complement. We keep using the notation $N[u] = [u^-, u^+]$ w.r.t. $\prec$. Parts 3 and 4 of the next proposition will be needed for proving Lemma 11.

▶ **Proposition 6.** *Let $\prec$ be any CA order of $\mathcal{N}[G]$, where G is a PCA graph with non-bipartite complement, and let $u, v \in V(G)$.*

1. *$N[u] \neq V(G)$ and u divides $N[u] = [u^-, u^+]$ into two cliques $[u^-, u]$ and $[u, u^+]$ of G.*
2. *$v \in [u, u^+]$ if and only if $u \in [v^-, v]$.*
3. *If $v \in [u, u^+]$, then $v^- \in [u^-, u]$ and $u^+ \in [v, v^+]$.*
4. *If $v \in [u, u^+]$ and $u \prec v$, then u^-, v^-, u, v, u^+, and v^+ occur under the order $\prec$ exactly in this circular sequence, where some of the neighboring vertices except u^- and v^+ may coincide.*

Proof of Lemma 5. Call an orientation of a graph G *round* if there is a CA order $\prec$ of $\mathcal{N}[G]$, such that each vertex v separates the arc $N[v] = [v^-, v^+]$ of $(V(G), \prec)$ into two arcs $[v^-, v)$ and $(v, v^+]$ such that the former consists of the in-neighbors of v and the latter consists of the out-neighbors of v. We also call such an orientation *compatible* with $\prec$.

As stated in [11, Corollary 2.9], any twin-free connected PCA graph with non-bipartite complement has a unique round orientation (up to reversal). Therefore, in order to prove that $\mathcal{N}[G]$ has a unique CA order (up to reversal), it suffices to show that any CA order $\prec$ of $\mathcal{N}[G]$ determines a compatible round orientations of G and that this correspondence is injective. Indeed, orienting an edge $\{u, v\}$ of G as (u, v) if $u \in [v^-, v)$ and as (v, u) if $u \in (v, v^+]$ is well-defined, since by Proposition 6, G has no universal vertex and $v \in [u, u^+]$ if and only if $u \in [v^-, v]$. Further, it is not hard to see that different CA orders produce different orientations. ◀

We close this section by giving a characterization of concave-round graphs G with bipartite complement using properties of $\mathcal{N}(\overline{G})$. Given a bipartite graph H and a bipartition $V(H) = U \cup W$ of its vertices into two independent sets, by $\mathcal{N}_U(H)$ we denote the hypergraph $\{N(w)\}_{w \in W}$ on the vertex set U. Note that $(\mathcal{N}_U(H))^* \cong \mathcal{N}_W(H)$. A bipartite graph H is called *convex* if its vertex set admits splitting into two independent sets U and W, such that $\mathcal{N}_U(H)$ is an interval hypergraph. If both $\mathcal{N}_U(H)$ and $\mathcal{N}_W(H)$ are interval hypergraphs, H is called *biconvex*; see, e.g., [28]. As G is co-bipartite concave-round if and only if its complement $H = \overline{G}$ is bipartite convex-round, the following fact gives the desired characterization.

▶ **Proposition 7** (implicitly in Lemma 3 of [29])**.** *A graph H is bipartite convex-round if and only if it is biconvex and if and only if $\mathcal{N}(H)$ is an interval hypergraph.*

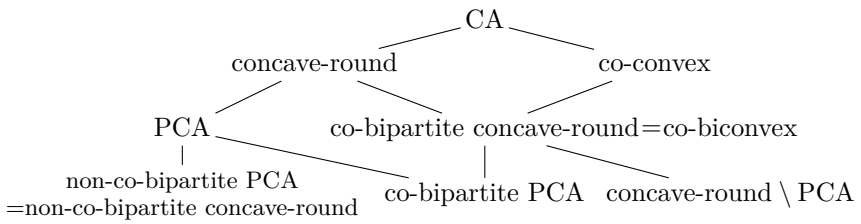


Figure 1 Inclusion structure of the classes of graphs under consideration.

5 Canonical arc representations of concave-round and PCA graphs

We are now ready to present our canonical representation algorithm for concave-round and PCA graphs.

▶ **Theorem 8.** *There is a logspace algorithm that solves the canonical arc representation problem for the class of concave-round graphs. Moreover, this algorithm outputs a proper arc representation whenever the input graph is PCA.*

For any class of intersection graphs, a canonical representation algorithm readily implies a canonical labeling algorithm of the same complexity. Vice versa, a canonical representation algorithm readily follows from a canonical labeling algorithm *and* a representation algorithm (not necessarily a canonical one). Proving Theorem 8 according to this scheme, we split our task in two parts: We first compute a canonical labeling λ of the input graph G and then we compute an arc representation α of the canonical form $\lambda(G)$. Then the composition $\alpha \circ \lambda$

is a canonical arc representation of G. As twins can be easily re-inserted in a (proper) arc representation, it suffices to compute α for the twin-free version of $\lambda(G)$, where in each twin-class we only keep one vertex.

We distinguish two cases depending on whether $\overline{G}$ is bipartite; see Fig. 1 for an overview of the involved graph classes.

Non-co-bipartite concave-round graphs. As mentioned above, any concave-round graph G whose complement is not bipartite is actually a PCA graph [29]. Hence, we have to compute a proper arc representation in this case.

Canonical labeling. We first transform G into its twin-free version G', where we only keep one vertex in each twin-class. Let n be the number of vertices in G'. We use the algorithm given by Theorem 1 to compute an arc representation ρ' of $\mathcal{N}[G']$. By Lemma 5, $\mathcal{N}[G']$ has a CA order which is unique up to reversing. Hence, in order to determine a canonical labeling of G, it suffices to consider the $2n$ arc representations $\rho_1, \ldots, \rho_{2n}$ of $\mathcal{N}[G]$ that can be obtained from ρ' by cyclic shifts and reversing and by re-inserting all the removed twins. As a canonical labeling ρ_i of G, we appoint one of these $2n$ variants that gives the lexicographically least canonical form $\rho_i(G)$ of G.

Proper arc representation. As mentioned above, it suffices to find such a representation for the twin-free graph G'. The arc representation ρ' of $\mathcal{N}[G']$ that we have already computed provides us with a CA order $\prec$ for $\mathcal{N}[G']$. By Lemmas 4 and 5, there is a proper arc representation $\alpha\colon V(G') \to \mathcal{A}$ of G' such that $\prec$ coincides with the associated geometric order $\prec_\alpha$. In order to construct α from $\prec$, we can assume that no two arcs $\alpha(v) = [a_v^-, a_v^+]$ and $\alpha(u) = [a_u^-, a_u^+]$ in $\mathcal{A}$ share an extreme point and that $V(\mathcal{A})$ consists of exactly $2n$ points. To determine a suitable circular order on $V(\mathcal{A})$, order the start points a_v^- according to $\prec$. Then there is a unique way to place the end points a_v^+ to make α a proper arc representation of G'. A careful implementation shows that this computation can be done in logspace.

Co-bipartite concave-round graphs. By Proposition 7, co-bipartite concave-round graphs are precisely the co-biconvex graphs. In fact, all co-convex graphs are circular-arc (this is implicit in [29]) and we can actually compute a canonical arc representation for these graphs in logspace.

Canonical labeling. A canonical labeling algorithm for convex graphs, and hence also for co-convex graphs, is designed in [18].

(Proper) arc representation. We first recall Tucker's argument [29] showing that, if the complement of G is a convex graph, then G is CA. We can assume that $\overline{G}$ has no fraternal vertices as those would correspond to twins in G.

Let $V(G) = U \cup W$ be a partition of $\overline{G}$ into independent sets such that $\mathcal{N}_U(\overline{G})$ is an interval hypergraph. Let $u_1, \ldots, u_k$ be an interval order on U for $\mathcal{N}_U(\overline{G})$. We construct an arc representation α for G on the cycle $\mathbb{Z}_{2k+2}$ by setting $\alpha(u_i) = [i, i+k]$ for each $u_i \in U$ and $\alpha(w) = [j+k+1, i-1]$ for each $w \in W$, where $N_{\overline{G}}(w) = [u_i, u_j]$ and the subscript $\overline{G}$ means that the vertex neighborhood is considered in the complement of G. Note that $\alpha(w) = \mathbb{Z}_{2k+2} \setminus \bigcup_{u \in N_{\overline{G}}(w)} \alpha(u)$. In the case that $N_{\overline{G}}(w) = \emptyset$, we set $\alpha(w) = [0, k]$. By construction, all arcs $\alpha(u)$ for $u \in U$ share a point (even two, k and $k+1$), the same holds true for all $\alpha(w)$ for $w \in W$ (they share the point 0), and any pair $\alpha(u)$ and $\alpha(w)$ is intersecting if and only if u and w are adjacent in G. Thus, α is indeed an arc representation for G.

In order to compute α in logspace, it suffices to compute a suitable bipartition $\{U, W\}$ of $\overline{G}$ and an interval order of the hypergraph $\mathcal{N}_U(\overline{G})$ in logspace. Finding a bipartition

$\{U, W\}$ such that $\mathcal{N}_U(\overline{G})$ is an interval hypergraph can be done by splitting $\overline{G}$ into connected components $H_1, \ldots, H_k$ (using Reingold's algorithm [26]) and finding such a bipartition $\{U_i, W_i\}$ for each component H_i. By using the logspace algorithm of [18] we can actually compute interval orders of the hypergraphs $\mathcal{N}_{U_i}(H_i)$ which can be easily pasted together to give an interval order of $\mathcal{N}_U(\overline{G})$. Together with the canonical labeling algorithm this implies that the canonical arc representation problem for co-convex graphs and, in particular, for co-bipartite concave-round graphs is solvable in logspace.

It remains to show that for co-bipartite PCA graphs we can actually compute a proper arc representation in logspace. As above, we assume that G is twin-free. By Lemma 4, the hypergraph $\mathcal{N}[G]$ has a tight CA order $\prec$. We can compute $\prec$ in logspace by running the algorithm given by Theorem 1 on the tightened hypergraph $(\mathcal{N}[G])^{\Subset}$. Any tight CA order of $\mathcal{N}[G]$ is also a tight CA order of $\mathcal{N}(\overline{G})$. Let $V(G) = U \cup W$ be a bipartition of $\overline{G}$ into two independent sets. Note that the restriction of a tight CA order of $\mathcal{N}(\overline{G})$ to $\mathcal{N}_U(\overline{G})$ is a tight interval order of the interval hypergraph $\mathcal{N}_U(\overline{G})$. Retracing Tucker's construction of an arc representation α for a co-convex graph G in the case that the interval order of $\mathcal{N}_U(\overline{G})$ is tight, we see that α now gives us a tight arc model for G. Thus, we only have to convert α into a proper arc representation α'. Tucker [29] described such a transformation, and Chen [7] showed that it can be implemented in AC^1. It is not hard to see that it can even be done in logspace. This completes the proof of Theorem 8 and we have additionally proved the following corollary.

▶ **Corollary 9.** *The canonical arc representation problem for co-convex graphs is solvable in logspace.*

6 Solving the Star System Problem

In this section, we present logspace algorithms for the Star System Problem: Given a hypergraph $\mathcal{H}$, find a graph G in a specified class of graphs C such that $\mathcal{N}[G] = \mathcal{H}$ (if such a graph exists). The term *star* refers to the closed neighborhood of a vertex in G. In this terminology, the problem is to identify the center of each star H in the star system $\mathcal{H}$. To denote this problem, we use the abbreviation *SSP*. Note that a logspace algorithm $\mathcal{A}$ solving the SSP for a class C cannot be directly used for solving the SSP for a subclass C' of C. For example, if $\mathcal{A}$ on input $\mathcal{H}$ outputs a solution G in $\mathsf{C} \setminus \mathsf{C}'$, then we don't know whether there is another solution G' in C'. However, if the SSP for C has unique solutions and if membership in C' is decidable in logspace, then it is easy to convert $\mathcal{A}$ into a logspace algorithm $\mathcal{A}'$ solving the SSP for C'.

▶ **Theorem 10.**

1. *The SSP for PCA and for co-convex graphs is solvable in logspace.*
2. *If G is a co-convex graph, then $N[G] \cong N[H]$ implies $G \cong H$.*

The implication stated in Theorem 10.2 is known to be true also for concave-round graphs (Chen [8]). As a consequence, since concave-round graphs form a logspace decidable subclass of the union of PCA and co-convex graphs, we can also solve the SSP for concave-round graphs in logspace.

The proof of Theorem 10 is outlined in the rest of this section. We design logspace algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ solving the SSP for non-co-bipartite PCA graphs and for co-convex graphs, respectively. Since by Theorem 10.2, the output of $\mathcal{A}_2$ is unique up to isomorphism, we can easily combine the two algorithms to obtain a logspace algorithm $\mathcal{A}_3$ solving the SSP

for all PCA graphs: On input $\mathcal{H}$ run $\mathcal{A}_1$ and $\mathcal{A}_2$ and check if one of the resulting graphs is PCA (recall that co-bipartite PCA graphs are co-convex; see Fig. 1).

Clearly, it suffices to consider the case that the input hypergraph $\mathcal{H}$ is connected.

Non-co-bipartite PCA graphs. Let $\mathcal{H}$ be the given input hypergraph and assume that $\mathcal{H} = \mathcal{N}[G]$ for a PCA graph G. By Theorem 3, $\mathcal{H}$ has to be a tight CA hypergraph, a condition that can be checked by testing if the tightened hypergraph $\mathcal{H}^{\Subset}$ is CA. Since G is concave-round, Proposition 7 implies that G is co-bipartite if and only if $\mathcal{N}(\overline{G}) = \overline{\mathcal{H}}$ is an interval hypergraph. It follows that the SSP on $\mathcal{H}$ can only have a non-co-bipartite PCA graph as solution if $\mathcal{H}^{\Subset}$ is CA and $\overline{\mathcal{H}}$ is not interval. Both conditions can be checked in logspace using the algorithms given by Theorem 1 and [18]. Further, it follows by Theorem 3 and Proposition 7 that in this case any SSP solution for $\mathcal{H}$ is a non-co-bipartite PCA graph (which is also connected because $\mathcal{H}$ is assumed to be connected).

By considering the quotient hypergraph with respect to twin-classes, we can additionally assume that $\mathcal{H}$ is twin-free.

In order to reconstruct G from $\mathcal{H}$, we have to choose the center in each star $H \in \mathcal{H}$. The following lemma considerably restricts this choice.

▶ **Lemma 11.** *Let G be a connected, non-co-bipartite and twin-free PCA graph and let $\prec$ be a circular order on $V(G)$ that is a CA order of $\mathcal{N}[G]$. Then $u \prec v$ holds exactly when $N[u] \prec_{\mathcal{N}[G]} N[v]$, where $\prec_{\mathcal{N}[G]}$ is the circular order on $\mathcal{N}[G]$ lifted from $\prec$.*

Proof. It suffices to show that $u \prec v$ implies $N[u] = [u^-, u^+] \prec_{\mathcal{N}[G]} N[v] = [v^-, v^+]$. To this end we show that there is no third vertex w such that the arcs $N[u]$, $N[w]$, and $N[v]$ appear in this sequence under the circular order $\prec_{\mathcal{N}[G]}$.

Suppose first that u and v are adjacent. Then it follows from Proposition 6.4, that the vertices u^-, v^-, u, v, u^+, and, v^+ appear in this circular sequence; see Fig. 2(a). We split our analysis into three cases, depending on the position of w on the cycle $(V(G), \prec)$. If $w \in (v, v^+]$, then Proposition 6.3 implies that $w^- \in [v^-, v]$ and $v^+ \in [w, w^+]$. If $w^- \neq v^-$, then $N[u]$, $N[v]$, and $N[w]$ appear in this sequence under $\prec_{\mathcal{N}[G]}$. The same holds true if $w^- = v^-$ because then the arc $[w^-, w^+]$ has to be longer than the arc $[v^-, v^+]$ (note that, if also $u^- = v^-$, then $[u^-, u^+]$ is shorter than $[v^-, v^+]$). The case that $w \in [u^-, u)$ is similar. If $w \in (v^+, u^-)$, then $w^- \in (v, u^-)$, and again $N[w]$ cannot be intermediate.

Figure 2 The two cases in the proof of Lemma 11.

Suppose now that u and v are not adjacent. It follows that $N[u] = [u^-, u]$ and $N[v] = [v, v^+]$; see Fig. 2(b). By Proposition 6.1, both $N[u]$ and $N[v]$ are cliques. Again we have to show that for no third vertex w, the arcs $N[u]$, $N[w]$, and $N[v]$ appear in this sequence under $\prec_{\mathcal{N}[G]}$. This is clear if $w^- \in (v, u^-)$. This is also so if $w^- = v$, because then the arc $[v, v^+]$, being a clique in G, must be shorter than the arc $[w^-, w^+]$. Finally we show that the remaining case $w^- \in [u^-, v)$ is not possible. Indeed, in this case $v \notin N[w]$, for else the non-adjacent vertices u and v would belong to the clique $[w, w^+]$. Hence, it would follow that $N[w] = [w^-, w^+] \subsetneq [u^-, u^+] = N[u]$, contradicting the fact that $N[u]$ is a clique. ◀

Lemma 11 states that the mapping $v \mapsto N[v]$ is an isomorphism between the two directed cycles $(V(G), \prec)$ and $(\mathcal{N}[G], \prec_{\mathcal{N}[G]})$. Since there are exactly n such isomorphisms, we get exactly n candidates $f_1, \ldots, f_n$ for the mapping $v \mapsto N[v]$. Hence, all we have to do is to use the algorithm given by Theorem 1 to compute a CA order $\prec$ of $\mathcal{H}$ and the corresponding lifted order $\prec_{\mathcal{H}}$ in logspace. Now for each isomorphism f between $(V(G), \prec)$ and $(\mathcal{H}, \prec_{\mathcal{H}})$ we have to check if selecting v as the center of the star $f(v)$ results in a graph G, that is, if for all $v, u \in V(G)$ it holds that $v \in f(v)$ and $v \in f(u) \Leftrightarrow u \in f(v)$.

Co-convex graphs. Let $\mathcal{H}$ be the given hypergraph and assume that $\mathcal{H} = \mathcal{N}[G]$ for a co-convex graph G. To facilitate the exposition, we also assume that the bipartite complement $\overline{G}$ is connected, with vertex partition U, W. Then $\overline{\mathcal{H}} = \mathcal{N}(\overline{G}) = \mathcal{N}_U(\overline{G}) \cup \mathcal{N}_W(\overline{G})$, where the vertex-disjoint hypergraphs $\mathcal{U} = \mathcal{N}_U(\overline{G})$ and $\mathcal{W} = \mathcal{N}_W(\overline{G})$ are dual (i.e., $\mathcal{U}^* \cong \mathcal{W}$), both connected, and at least one of them is interval, say, $\mathcal{U}$. We need a simple auxiliary fact.

▶ **Proposition 12.** *Let H be a graph without isolated vertices and let $\mathcal{L}$ be a connected component of $\mathcal{N}(H)$. Denote $U = V(\mathcal{L})$. Then either U is an independent set in H or U spans a connected component of H. Moreover, if U is independent, then there is a connected component of H that is a bipartite graph with U being one of its vertex classes.*

Denote $\mathcal{K} = \overline{\mathcal{H}}$ and assume that $\mathcal{K} = \mathcal{N}(H)$ for some graph H. Note that H cannot have an isolated vertex. Proposition 12 implies that either H is a connected bipartite graph with partition U, W or H has two connected components H_1 and H_2 with $V(H_1) = U$ and $V(H_2) = W$. However, the second possibility leads to a contradiction. Indeed, since the hypergraph $\mathcal{N}(H_1) = \mathcal{U}$ is interval, Proposition 7 implies that H_1 is bipartite, contradicting the connectedness of $\mathcal{U}$. Therefore, H must be connected and bipartite with vertex partition U, W.

Recall that the *incidence graph* of a hypergraph $\mathcal{X}$ is the bipartite graph with vertex classes $V(\mathcal{X})$ and $\mathcal{X}$ where $x \in V(\mathcal{X})$ and $X \in \mathcal{X}$ are adjacent if $x \in X$ (if X has multiplicity k in $\mathcal{X}$, it contributes k fraternal vertices in the incidence graph). Since H is isomorphic to the incidence graph of the hypergraph $\mathcal{U}$ (as well as $\mathcal{W}$), H is logspace reconstructible from $\mathcal{K}$ up to isomorphism and, in particular, $H \cong \overline{G}$. Thus, the solution to the SSP on $\mathcal{H}$ is unique up to isomorphism.

After these considerations we are ready to describe our logspace algorithm for solving the SSP for the class of co-convex co-connected graphs. Given a hypergraph $\mathcal{H}$, we first check if $\overline{\mathcal{H}}$ has exactly two connected components, say $\mathcal{U}$ and $\mathcal{W}$. This can be done by running Reingold's algorithm for the connectivity problem [26] on the intersection graph $\mathbb{I}(\overline{\mathcal{H}})$. If this is not the case, there is no solution in the desired class. Otherwise, we construct the incidence graph H of the hypergraph $\mathcal{U}$ (or of $\mathcal{W}$, which should give the same result up to isomorphism) and take its complement $\overline{H}$. Note that this works well even if $\overline{H}$ has twins: the twins in $V(\mathcal{U})$ are explicitly present, while the twins in $V(\mathcal{W})$ are represented by multiple hyperedges in $\mathcal{U}$.

As argued above, if the SSP on $\mathcal{H}$ has a co-convex co-connected solution, then the closed neighborhood hypergraph $\mathcal{H}' = \mathcal{N}[\overline{H}]$ of $\overline{H}$ is isomorphic to $\mathcal{H}$. However, it may not be equal to $\mathcal{H}$. In this case we compute an isomorphism φ from $\mathcal{H}'$ to $\mathcal{H}$ or, the same task, from $\overline{\mathcal{H}'}$ to $\overline{\mathcal{H}}$. This can be done by the algorithms of [18] and Corollary 2, because at least one of the connected components of $\overline{\mathcal{H}'} \cong \overline{\mathcal{H}}$ is an interval hypergraph and the other component is isomorphic to the dual of an interval hypergraph. Now, the isomorphic image $G = \varphi(\overline{H})$ of $\overline{H}$ is the desired solution to the SSP on $\mathcal{H}$ as $\mathcal{N}[\varphi(\overline{H})] = \varphi(\mathcal{N}[\overline{H}]) = \mathcal{H}$.

Acknowledgement. We thank Bastian Laubner for useful discussions at the early stage of this work.

References

1 M. Aigner and E. Triesch. Reconstructing a graph from its neighborhood lists. *Combinatorics, Probability & Computing*, 2:103–113, 1993.

2 F. S. Annexstein and R. P. Swaminathan. On testing consecutive-ones property in parallel. *Discrete Applied Mathematics*, 88(1-3):7–28, 1998.

3 J. Bang-Jensen, J. Huang, and A. Yeo. Convex-round and concave-round graphs. *SIAM J. Discrete Math.*, 13(2):179–193, 2000.

4 K. Booth, G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976.

5 E. Boros, V. Gurvich, and I. E. Zverovich. Neighborhood hypergraphs of bipartite graphs. *Journal of Graph Theory*, 58(1):69–95, 2008.

6 L. Chen. Efficient parallel recognition of some circular arc graphs, I. *Algorithmica*, 9(3):217–238, 1993.

7 L. Chen. Efficient parallel recognition of some circular arc graphs, II. *Algorithmica*, 17(3):266–280, 1997.

8 L. Chen. Graph isomorphism and identification matrices: Parallel algorithms. *IEEE Trans. Parallel Distrib. Syst.*, 7(3):308–319, 1996.

9 L. Chen and Y. Yesha. Parallel recognition of the consecutive ones property with applications. *J. Algorithms*, 12(3):375–392, 1991.

10 A.R. Curtis, M.C. Lin, R.M. McConnell, Y. Nussbaum, F.J. Soulignac, J.P. Spinrad, J.L. Szwarcfiter. Isomorphism of graph classes related to the circular-ones property. E-print: *http://arxiv.org/abs/1203.4822*, 2012.

11 X. Deng, P. Hell, J. Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Comput.*, 25:390–403, 1996.

12 F. V. Fomin, J. Kratochvíl, D. Lokshtanov, F. Mancini, and J. A. Telle. On the complexity of reconstructing H-free graphs from their Star Systems. *Journal of Graph Theory*, 68(2):113–124, 2011.

13 F. Gavril, R. Y. Pinter, and S. Zaks. Intersection representations of matrices by subtrees and unicycles on graphs. *Journal of Discrete Algorithms*, 6(2):216–228, 2008.

14 W.-L. Hsu. $O(MN)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM J. Comput.*, 24(3):411–439, 1995.

15 W.-L. Hsu. A simple test for the consecutive ones property. *J. Algorithms*, 43(1):1–16, 2002.

16 W.-L. Hsu and R. M. McConnell. PC trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):99–116, 2003.

17 H. Kaplan, Y. Nussbaum. Certifying algorithms for recognizing proper circular-arc graphs and unit circular-arc graphs. *Discr. Appl. Math.*, 157:3216–3230, 2009.

18 J. Köbler, S. Kuhnert, B. Laubner, and O. Verbitsky. Interval graphs: Canonical representations in Logspace. *SIAM J. on Computing*, 40(5):1292–1315, 2011.

19 J. Köbler, S. Kuhnert, and O. Verbitsky. Solving the canonical representation and star system problems for proper circular-arc graphs in logspace. E-print: *http://arxiv.org/abs/1202.4406*, 2012.

20 J. Köbler, S. Kuhnert, and O. Verbitsky. Around and beyond the isomorphism problem for interval graphs. *Bulletin of the EATCS*, 107:44–71, 2012.

21 F. Lalonde. Le probleme d'etoiles pour graphes est NP-complet. *Discrete Mathematics*, 33(3):271–280, 1981.

22 M. C. Lin, F. J. Soulignac, and J. L. Szwarcfiter. A simple linear time algorithm for the isomorphism problem on proper circular-arc graphs. *Proc. 11th SWAT*, LNCS vol. 5124, pp. 355–366, 2008.

23 G. Lueker and K. Booth. A linear time algorithm for deciding interval graph isomorphism. *J. ACM*, 26(2):183–195, 1979.

24 R. M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.

25 A. Ouangraoua, A. Bergeron, and K. M. Swenson. Theory and practice of ultra-perfection. *J. Comp. Bio.*, 18(9): 1219-1230, 2011.

26 O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

27 F. Roberts. Indifference graphs. Proof Tech. Graph Theory, Proc. 2nd Ann Arbor Graph Theory Conf. 1968, 139–146, 1969.

28 J. Spinrad. *Efficient graph representations.* Fields Institute Monographs, 19. AMS, 2003.

29 A. Tucker. Matrix characterizations of circular-arc graphs. *Pac. J. Math.*, 39:535–545, 1971.

Directed Acyclic Subgraph Problem Parameterized above the Poljak-Turzík Bound

Robert Crowston[1], Gregory Gutin[1], and Mark Jones[1]

1 **Royal Holloway, University of London**
Egham TW20 0EX, UK
[robert,gutin,markj]@cs.rhul.ac.uk

Abstract

An oriented graph is a directed graph without directed 2-cycles. Poljak and Turzík (1986) proved that every connected oriented graph G on n vertices and m arcs contains an acyclic subgraph with at least $\frac{m}{2} + \frac{n-1}{4}$ arcs. Raman and Saurabh (2006) gave another proof of this result and left it as an open question to establish the parameterized complexity of the following problem: does G have an acyclic subgraph with least $\frac{m}{2} + \frac{n-1}{4} + k$ arcs, where k is the parameter? We answer this question by showing that the problem can be solved by an algorithm of runtime $(12k)!n^{O(1)}$. Thus, the problem is fixed-parameter tractable. We also prove that there is a polynomial time algorithm that either establishes that the input instance of the problem is a Yes-instance or reduces the input instance to an equivalent one of size $O(k^2)$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Acyclic Subgraph, Fixed-parameter tractable, Polynomial Kernel

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.400

1 Introduction

The problem of finding the maximum acyclic subgraph in a directed graph[1] is well-studied in the literature in graph theory, algorithms and their applications alongside its dual, the feedback arc set problem, see, e.g., Chapter 15 in [2] and references therein. This is true, in particular, in the area of parameterized algorithmics [3, 11, 12, 19].

Each directed graph D with m arcs has an acyclic subgraph with at least $m/2$ arcs. To obtain such a subgraph, order the vertices $x_1, \ldots, x_n$ of D arbitrarily and consider two spanning subgraphs of D: D' with arcs of the form x_ix_j, and D'' with arcs of the form x_jx_i, where $i < j$. One of D' and D'' has at least $m/2$ arcs. Moreover, $m/2$ is the largest size of an acyclic subgraph in every symmetric digraph S (in a symmetric digraph the existence of an arc xy implies the existence of an arc yx). Thus, it makes sense to consider the parameterization[2] above the tight bound $m/2$: decide whether a digraph D contains an acyclic subgraph with at least $m/2 + k$ arcs, where k is the parameter. Mahajan *et al.* [14] and Raman and Saurabh [19] asked what the complexity of this problem is. For the case of oriented graphs (i.e., directed graphs with no directed cycles of length 2), Raman and Saurabh [19] proved that the problem is fixed-paramter tractable. A generalization of this

[1] We use standard terminology and notation on directed graphs which almost always follows [2]. Some less standard and this-paper-specific digraph terminology and notation is provided in the end of this section.

[2] We use standard terminology on parameterized algorithmics, see, e.g., [7, 8, 17].

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 400–411

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

problem to integer-arc-weighted digraphs (where $m/2$ is replaced by the half of the total weight of D) was proved to be fixed-parameter tractable in [11].

For oriented graphs, $m/2$ is no longer a tight lower bound on the maximum size of an acyclic subgraph. Poljak and Turzík [18] proved the following tight bound on the maximum size of an acyclic subgraph of a connected oriented graph D: $\frac{m}{2} + \frac{n-1}{4}$. To see that the bound is indeed tight consider a directed path $x_1x_2 \dots x_{2t+1}$ and add to it arcs $x_3x_1, x_5x_3, \dots, x_{2t+1}x_{2t-1}$. This oriented graph H_t consists of t directed 3-cycles and has $2t+1$ vertices and $3t$ arcs. Thus, $\frac{m}{2} + \frac{n-1}{4} = 2t$ and $2t$ is the maximum size of an acyclic subgraph of H_t: we have to delete an arc from every directed 3-cycle as the cycles are arc-disjoint.

Raman and Saurabh [19] asked to determine the parameterized complexity of the following problem: decide whether a connected oriented graph D has an acyclic subgraph with at least $\frac{m}{2} + \frac{n-1}{4} + k$ arcs, where k is the parameter. Answering this question, we will prove that this problem is fixed-parameter tractable and admits a kernel with $O(k^2)$ vertices and $O(k^2)$ arcs.

Observe that we may replace k by $\frac{k}{4}$ to ensure that the parameter k is always integral. Therefore, the complexity of the Raman-Saurabh problem above is equivalent to that of the following parameterized problem.

Acyclic Subgraph above Poljak-Turzík Bound (ASAPT)
Instance: An oriented connected graph G with n vertices and m arcs.
Parameter: k.
Question: Does G contain an acyclic subgraph with at least $\frac{m}{2} + \frac{n-1}{4} + \frac{k}{4}$ arcs?

Just a few years ago, as recorded by Mahajan *et al.* [14], there were only very few sporadic results on problems parameterized above or below nontrivial tight bounds. By now the situation has changed quite dramatically: most of the open questions in [14] on parameterized complexity of problems parameterized above or below tight bounds have been solved. In the process of solving these problems, some methods and approaches have been developed. One such method is the use of lower bounds on the maximum value of a pseudo-boolean function. The lower bounds are obtained using either a combination of probabilistic arguments and Fourier analysis inequalities [1, 9, 10, 11] or a combination of linear algebraic, algorithmic and combinatorial results and approaches [4]. Unfortunately, this method appears to be applicable mainly to constraint satisfaction problems rather than those on graphs and, thus, development of other methods applicable to problems on graphs parameterized above or below tight bounds, is of great interest. Recently, such a method based on linear programming was investigated in [6, 16].

This paper continues development of another such method, which is a combination of structural graph-theoretical and algorithmic approaches, recently introduced in [5]; in fact, this paper demonstrates that the approach of [5] for designing a fixed-parameter algorithm and producing a polynomial-size kernel for a problem on undirected graphs parameterized above tight bound can be modified to achieve the same for a problem on directed graphs.

In a nutshell, the method uses both two-way reduction rules (i.e., rules reducing an instance to an equivalent one) and one-way reduction rules (in such a rule if the reduced instance is a Yes-instance, then the original instance is also a Yes-instance) to transform the input instance to a trivial graph. If the reduction rules do not allow us to conclude that the input instance is a Yes-instance, then the input instance has a relatively "regular" structure that can be used to solve the problem by a fixed-parameter dynamic programming

algorithm. To establish the reduction rules and to show their "completeness", a structural result on undirected graphs is used, such as Lemma 8 in this paper or Lemma 3 in [5].

While the underlying approach in both papers is the same, the proofs used are different due to the specifics of each problem. In particular, a different set of reduction rules is used, and the "regular" structure derived in our paper is rather different from that in [5]. The dynamic programming algorithm and kernel proof are also completely different, other than the fact that in both papers the proofs are based on the "regular" structure of the graph. Finally, note that whilst the kernel obtained in [5] has $O(k^5)$ vertices, we obtain a kernel with just $O(k^2)$ vertices and $O(k^2)$ arcs.

The paper is organized as follows. In the next section,we obtain two basic results on oriented graphs. Two-way and one-way reduction rules are introduced in Sections 3 and 4, respectively. Fixed-parameter tractability of ASAPT is proved in Section 5. Section 6 is devoted to proving the existence of a polynomial kernel. In Section 7, we briefly mention another recent paper that showed that ASAPT is fixed-parameter tractable. We also discuss two open questions.

Some Digraph Terminology and Notation. Let D be a directed graph on n vertices and m arcs. For a vertex x in D, the *out-degree* $d^+(x)$ is the number of arcs of D leaving x and the *in-degree* $d^-(x)$ is the number of arcs of D entering x. For a subset S of vertices of D, let $d^+(S)$ denote the number of arcs of D leaving S and $d^-(S)$ the number of arcs of D entering S. For subsets A and B of vertices of D, let $E(A,B)$ denote the set of arcs with exactly one endpoint in each of A and B (in both directions). For a set S of vertices, $D[S]$ is the subgraph of D induced by S. When $S=\{s_1,\ldots,s_p\}$, we will write $D[s_1,\ldots,s_p]$ instead of $D[\{s_1,\ldots,s_p\}]$. The *underlying graph* $\mathrm{UN}(D)$ of D is the undirected graph obtained from D by replacing all arcs by edges with the same end-vertices and getting rid of one edge in each pair of parallel edges. The *connected components* of D are connected components of $\mathrm{UN}(D)$; D is *connected* if $\mathrm{UN}(D)$ is connected. Vertices x and y of D are *neighbors* if there is an arc between them. The maximum number of arcs in an acyclic subgraph of D will be denoted by $\mathrm{a}(D)$. Let $\gamma(D)=\frac{m}{2}+\frac{n-c}{4}$, where c is the number of connected components of D. By the Poljak-Turzík bound, we have

$$\mathrm{a}(G)\geq\gamma(G) \tag{1}$$

for every oriented graph G. A *tournament* is an oriented graph obtained from a complete graph by orienting its edges arbitrarily. A *directed p-cycle* is a directed cycle with p arcs.

2 Basic Results on Oriented Graphs

In our arguments we use the following simple correspondence between acyclic digraphs and orderings of vertices in digraphs. Let H be an acyclic spanning subgraph of a digraph D. It is well-known [2] and easy to see that there is an ordering $x_1,\ldots,x_n$ of vertices of D such that if x_ix_j is an arc of H then $i<j$. On the other hand, any ordering $x_1,\ldots,x_n$ of vertices of a digraph $D=(V,A)$ leads to an acyclic spanning subgraph of D: consider the subgraph induced by $\{x_ix_j : x_ix_j\in A, i<j\}$. As we study maximum-size acyclic subgraphs, we may restrict ourselves to acyclic spanning subgraphs. Thus, we may use interchangeably the notions of acyclic spanning subgraphs and vertex orderings.

There are some known lower bounds on $\mathrm{a}(T)$ for tournaments T on n vertices, see, e.g., [20] and references therein. We show the following useful bound which we were unable to find in the literature.

▶ **Lemma 1.** *For a tournament T on n vertices with $m = \binom{n}{2}$ arcs, we can, in polynomial time, find an acyclic subgraph with at least $\frac{m}{2} + \frac{3n}{4} - 1 = \gamma(T) + \frac{2n-3}{4}$ arcs, if n is even, or $\frac{m}{2} + \frac{3(n-1)}{4} - 1 = \gamma(T) + \frac{2n-6}{4}$ arcs, if n is odd.*

Proof. We prove the lemma by induction. The claim can easily be checked for $n = 1$ and $n = 2$ and we may assume that $n \geq 3$.

Consider first the case when n is even. Suppose that there exists a vertex x such that $d^+(x) \geq \frac{n}{2} + 1$. Consider the tournament $T' = T - x$, with $m' = m - (n-1)$ arcs and $n' = n - 1$ vertices. By induction, there is an ordering on T' that produces an acyclic spanning subgraph H' of T' such that

$$\mathrm{a}(H') \geq \frac{m'}{2} + \frac{3(n'-1)}{4} - 1 = \frac{m-(n-1)}{2} + \frac{3(n-2)}{4} - 1 = \frac{m}{2} + \frac{3n}{4} - \frac{n}{2} - 2.$$

Now add x to the beginning of this ordering. This produces an acyclic spanning subgraph H of T such that $\mathrm{a}(H) \geq \mathrm{a}(H') + \frac{n}{2} + 1 \geq \frac{m}{2} + \frac{3n}{4} - 1$.

If there is a vertex x such that $d^-(x) \geq \frac{n}{2} + 1$, the same argument applies, but x is added to the end of the ordering.

Otherwise, for every vertex x of T, $d^+(x) \in \{\frac{n}{2} - 1, \frac{n}{2}\}$. Moreover, by considering the sum of out-degrees, exactly half the vertices have out-degree $\frac{n}{2}$. Hence, if $n \geq 4$, there are at least two vertices with out-degree $\frac{n}{2}$. Let x and y be two such vertices, and suppose, without loss of generality, that there is an arc from x to y. Now consider $T' = T - \{x, y\}$ with $m' = m - (2n-3)$ edges and $n' = n - 2$ vertices. By induction, there is an ordering on the vertices of T' that produces an acyclic subgraph with at least $\frac{m'}{2} + \frac{3n'}{4} - 1 = \frac{m}{2} + \frac{3n}{4} - n - 1$ arcs. Place x and y at the beginning of this ordering, with x occurring before y. Then this will add all the arcs from x and y to the acyclic subgraph. Thus, $\mathrm{a}(T) \geq \frac{m}{2} + \frac{3n}{4} - n - 1 + n = \frac{m}{2} + \frac{3n}{4} - 1$.

Now suppose that n is odd. Let x be any vertex in T, and let $T' = T - x$. By induction, there is an ordering on T' that produces an acyclic subgraph with at least $\frac{m'}{2} + \frac{3n'}{4} - 1$ arcs, where $n' = n - 1$ is the number of vertices and $m' = m - (n-1)$ is the number of arcs in T'. By placing x either at the beginning or end of this ordering, we may add at least $(n-1)/2$ arcs. Thus, $\mathrm{a}(T) \geq \frac{m-(n-1)}{2} + \frac{3(n-1)}{4} - 1 + \frac{n-1}{2} = \frac{m}{2} + \frac{3(n-1)}{4} - 1$. ◀

▶ **Lemma 2.** *Let S be a nonempty set of vertices of an oriented graph G such that both $G - S$ and $G[S]$ are connected. If $\mathrm{a}(G - S) \geq \gamma(G - S) + \frac{k'}{4}$ and $\mathrm{a}(G[S]) \geq \gamma(G[S]) + \frac{k''}{4}$, then $\mathrm{a}(G) \geq \gamma(G) + \frac{k'+k''-1}{4} + \frac{|d^+(S)-d^-(S)|}{2}$. In particular, $\mathrm{a}(G) \geq \gamma(G) + \frac{k'+k''-1}{4}$ if $|E(S, V(G) \setminus S)|$ is even and $\mathrm{a}(G) \geq \gamma(G) + \frac{k'+k''+1}{4}$, if $|E(S, V(G) \setminus S)|$ is odd.*

Proof. Form an acyclic subgraph on G as follows. Assume without loss of generality that $d^+(S) \geq d^-(S)$. Pick the arcs leaving S together with the arcs of the acyclic subgraphs in $G - S$ and $G[S]$. This forms an acyclic subgraph H. Let $m = m' + m'' + \bar{m}$ and $n = n' + n''$, where $G - S$ has m' arcs and n' vertices, $G[S]$ has m'' arcs and n'' vertices and $\bar{m} = d^+(S) + d^-(S)$. The acyclic subgraph H has at least $\gamma(G-S) + \frac{k'}{4} + \gamma(G[S]) + \frac{k''}{4} + \frac{\bar{m}}{2} + \frac{d^+(S)-d^-(S)}{2} = \frac{m'+m''+\bar{m}}{2} + \frac{n'-1}{4} + \frac{n''-1}{4} + \frac{k'}{4} + \frac{k''}{4} + \frac{d^+(S)-d^-(S)}{2} = \gamma(G) + \frac{k'+k''-1}{4} + \frac{d^+(S)-d^-(S)}{2}$ arcs, as required. ◀

3 Two-way Reduction Rules

In the rest of this paper, G stands for an arbitrary connected oriented graph with n vertices and m arcs. We initially apply two 'two-way' reduction rules to (G, k) to form a new instance (G', k) such that (G', k) is a YES-instance of ASAPT if and only if (G, k) is a YES-instance

of ASAPT (i.e., the value of the parameter remains unchanged). We denote the number of vertices and arcs in G' by n' and m', respectively.

▶ Reduction Rule 1. Let x be a vertex and S a set of two vertices such that $G[S]$ is a component of $G - x$ and $G[S \cup \{x\}]$ is a directed 3-cycle. Then $G' := G - S$.

▶ **Lemma 3.** *If (G', k) is an instance obtained from (G, k) by an application of Rule 1, then G' is connected, and (G', k) is a* YES*-instance of* ASAPT *if and only if (G, k) is a* YES*-instance of* ASAPT.

Proof. Any two components of $G' - x$ will be connected by x and so G' is connected. Since $\mathrm{a}(G') = \mathrm{a}(G) - 2$, $m' = m - 3$ and $n' = n - 2$, we have $\mathrm{a}(G) \geq \frac{m}{2} + \frac{n-1}{4} + \frac{k}{4}$ if and only if $\mathrm{a}(G') \geq \frac{m'}{2} + \frac{n'-1}{4} + \frac{k}{4}$. ◀

▶ Reduction Rule 2. Let a, b, c, d, e be five vertices in G such that $G[a, b, c]$ and $G[c, d, e]$ are directed 3-cycles, $G[a, b, c, d, e] = G[a, b, c] \cup G[c, d, e]$ and a, e are the only vertices in $\{a, b, c, d, e\}$ that are adjacent to a vertex in $G - \{a, b, c, d, e\}$. To obtain G' from G, delete b, c and d, add a new vertex x and three arcs such that $G[a, x, e]$ is a directed 3-cycle.

▶ **Lemma 4.** *If (G', k) is an instance obtained from (G, k) by an application of Rule 2, then G' is connected, and (G', k) is a* YES*-instance of* ASAPT *if and only if (G, k) is a* YES*-instance of* ASAPT.

Proof. Clearly, G' is connected. Note that $\mathrm{a}(G') = \mathrm{a}(G) - 2$, $m' = m - 3$ and $n' = n - 2$. Thus, we have $\mathrm{a}(G) \geq \frac{m}{2} + \frac{n-1}{4} + \frac{k}{4}$ if and only if $\mathrm{a}(G') \geq \frac{m'}{2} + \frac{n'-1}{4} + \frac{k}{4}$. ◀

4 One-way Reduction Rules

Recall that G stands for an arbitrary connected oriented graph with n vertices and m arcs. We will apply reduction rules transforming an instance (G, k) of ASAPT into a new instance (G', k'), where G' is an oriented graph with n' vertices and m' arcs, and k' is the new value of the parameter. We will see that for the reduction rules of this section the following property will hold: if (G', k') is a YES-instance then (G, k) is a YES-instance, but not necessarily vice versa. Thus, the rules of this section are called one-way reduction rules.

▶ Reduction Rule 3. Let x be a vertex such that $G - x$ is connected, and $d^+(x) \neq d^-(x)$. To obtain (G', k') remove x from G and reduce k by $2|d^+(x) - d^-(x)| - 1$.

▶ **Lemma 5.** *If (G', k') is an instance reduced from (G, k) by an application of Rule 3, then G' is connected, and if (G', k') is a* YES*-instance then (G, k) is a* YES*-instance.*

Proof. Let (G', k') be a YES-instance. Then by Lemma 2 with $S = \{x\}$ and $k'' = 0$, $\mathrm{a}(G) \geq \gamma(G) + \frac{k'-1}{4} + \frac{|d^+(S) - d^-(S)|}{2} = \gamma(G) + \frac{k}{4}$, as required. ◀

▶ Reduction Rule 4. Let S be a set of vertices such that $G - S$ is connected, $G[S]$ is a tournament, and $|S| \geq 4$. To obtain (G', k'), remove S from G and reduce k by $2|S| - 4$ if S is even, or $2|S| - 7$ if $|S|$ is odd.

▶ **Lemma 6.** *If (G', k') is an instance obtained from (G, k) by an application of Rule 4, then G' is connected, and if (G', k') is a* YES*-instance then (G, k) is a* YES*-instance.*

Proof. Suppose $|S|$ is even. By Lemma 1, $\mathrm{a}(G[S]) \geq \gamma(G[S]) + \frac{2|S|-3}{4}$. By Lemma 2, if $\mathrm{a}(G') \geq \gamma(G') + (k - 2|S| + 4)/4$, then $\mathrm{a}(G) \geq \gamma(G) + \frac{(k-2|S|+4)+(2|S|-3)-1}{4} = \gamma(G) + \frac{k}{4}$, as required.

A similar argument applies in the case when $|S|$ is odd, except the bound from Lemma 1 is $\gamma(G[S]) + \frac{2|S|-6}{4}$, and so $k' = k - (2|S| - 7)$ is applied. ◀

▶ Reduction Rule 5. Let S be a set of three vertices such that the underlying graph of $G[S]$ is isomorphic to P_3, and $G - S$ is connected. To obtain (G', k'), remove S from G and reduce k by 1.

▶ **Lemma 7.** *If (G', k') is an instance obtained from (G, k) by an application of Rule 5, then G' is connected, and if (G', k') is a* YES*-instance then (G, k) is a* YES*-instance.*

Proof. Observe that $\mathrm{a}(G[S]) = \gamma(G[S]) + \frac{1}{2}$. Hence, by Lemma 2, if $\mathrm{a}(G') \geq \gamma(G') + (k-1)/4$, then $\mathrm{a}(G) \geq \gamma(G) + k/4$. ◀

5 Fixed-Parameter Tractability of ASAPT

The next lemma follows immediately from a nontrivial structural result of Crowston *et al.* (Lemma 3 in [5]).

▶ **Lemma 8.** *Given any connected undirected graph H, at least one of the following properties holds:*

- ***A*** *There exist $v \in V(H)$ and $X \subseteq V(H)$ such that X is a connected component of $H - v$ and X is a clique;*
- ***B*** *There exist $a, b, c \in V(H)$ such that $H[\{a, b, c\}]$ is isomorphic to P_3 and $H - \{a, b, c\}$ is connected;*
- ***C*** *There exist $x, y \in V(H)$ such that $\{x, y\} \notin E(H)$, $H - \{x, y\}$ is disconnected, and for all connected components X of $H - \{x, y\}$, except possibly one, $X \cup \{x\}$ and $X \cup \{y\}$ are cliques.*

▶ **Lemma 9.** *For any connected oriented graph G with at least one edge, one of Rules 1, 3, 4, 5 applies.*

Proof. If there is a vertex $x \in X$ such that $G - x$ is connected and $d^+(x) \neq d^-(x)$ (we will call such a case an *unbalanced case*), then Rule 3 applies. Thus, assume that for each $x \in X$ such that $G - x$ is connected we have $d^+(x) = d^-(x)$.

Consider the case when property A holds. If $|X| \geq 4$, Rule 4 applies on $S = X$. If $|X| = 3$, there has to be exactly one arc between X and v and $G[X]$ is a directed 3-cycle as otherwise we have an unbalanced case. Let $x \in X$ be the endpoint of this arc in X. Then Rule 1 applies with $S = X \backslash \{x\}$. If $|X| = 2$, then $G[X \cup \{v\}]$ is a directed 3-cycle (as otherwise we have an unbalanced case) and so Rule 1 applies. We cannot have $|X| = 1$ as this is an unbalanced case.

If property B holds, then Rule 5 can be applied to the path P_3 formed by a, b, c in the underlying graph of G.

Consider the case when property C holds. We may assume without loss of generality that the non-tournament component is adjacent to y.

Consider the subcase when $G - \{x, y\}$ has two connected components, X_1 and X_2, that are tournaments. Let $x_1 \in X_1$, $x_2 \in X_2$ and observe that the subgraph induced by x_1, x, x_2 forms a P_3 in the underlying graph of G and $G - \{x_1, x, x_2\}$ is connected, and so Rule 5 applies.

Now consider the subcase when $G - \{x, y\}$ has only one connected component X that is a tournament. If $|X| \geq 3$, then $X \cup \{x\}$ is a tournament with least four vertices, and so Rule 4 applies. If $|X| = 2$, then let $X = \{a, b\}$. Observe that a is adjacent to three vertices,

b, x, y, and so we have an unbalanced case to which Rule 3 applies. Finally, $X = \{a\}$ is a singleton, then observe that x, a, y form a P_3 in the underlying graph of G and $G - \{x, a, y\}$ is connected, and so Rule 5 applies. ◀

In this paper, we consider the one-vertex undirected graph as 2-connected. A maximal 2-connected induced subgraph of an undirected graph is called a *block*. An undirected graph H is called a *forest of cliques* if each block of H is a clique. A subgraph B of an oriented graph G is a *block* if $\mathrm{UN}(B)$ is a block in $\mathrm{UN}(G)$. An oriented graph G is a *forest of cliques* if $\mathrm{UN}(G)$ is a forest of cliques. A connected graph H that is a forest of cliques is known as a *tree of cliques*.

▶ **Lemma 10.** *Given a connected oriented graph G and integer k, we can either show that (G, k) is a* YES*-instance of* ASAPT*, or find a set U of at most $3k$ vertices such that $G - U$ is a forest of cliques with the following properties:*

1. *Every block in $G - U$ contains at most three vertices;*
2. *Every block X in $G - U$ with $|X| = 3$ induces a directed 3-cycle in G;*
3. *Every connected component in $G - U$ has at most one block X with $|X| = 2$ vertices;*
4. *There is at most one block in $G - U$ with one vertex (i.e., there is at most one isolated vertex in $G - U$).*

Proof. Apply Rules 1, 3, 4, 5 exhaustively, and let U be the set of vertices removed by Rules 3, 4, and 5 (but not Rule 1). If we reduce to an instance (G'', k'') with $k'' \leq 0$, then by Lemmas 3, 5, 6 and 7, (G, k) is a YES-instance and we may return YES. Now assume that, in the completely reduced instance (G'', k''), $k'' > 0$. We will prove that $|U| \leq 3k$ and $G - U$ satisfies the four properties of the lemma.

Observe that each time k is decreased by a positive integer q, at most $3q$ vertices are added to U. Thus, $|U| \leq 3k$. The rest of our proof is by induction. Observe that, by Lemma 9, for the completely reduced instance (G'', k'') either $G'' = \emptyset$ or G'' consists of a single vertex. Thus, $G'' - U$ satisfies the four properties of the lemma, which forms the basis of our induction.

For the induction step, consider an instance (G'', k'') obtained from the previous instance (G', k') by the application of a reduction rule. By the induction hypothesis, $G'' - U$ satisfies the four properties of the lemma. In the application of each of Rules 3, 4 and 5, the vertices deleted are added to U. Hence $G'' - U = G' - U$ and we are done unless G'' is obtained from G' by an application of Rule 1. Recall that in Rule 1 we delete a set S such that $G[S \cup \{x\}]$ forms a directed 3-cycle. We do not add S to U. If $x \in G'' - U$, then in $G' - U$, $S \cup \{x\}$ forms a block of size 3 that is a directed 3-cycle. If $x \notin G'' - U$, then in $G' - U$, S forms a new connected component with one block S with $|S| = 2$ vertices. Thus, $G' - U$ satisfies the four properties. ◀

▶ **Theorem 11.** *There is an algorithm for* ASAPT *of runtime $O((3k)!n^{O(1)})$.*

Proof. We may assume that for a connected oriented graph G we have the second alternative in the proof of Lemma 10, i.e., we are also given the set U of at most $3k$ vertices satisfying the four properties of Lemma 10. Consider an algorithm which generates all orderings of U, in time $O((3k)!)$ as $|U| \leq 3k$. An ordering $u_1, u_2, ..., u_{|U|}$ of U means that in the acyclic subgraph of G we are constructing, we keep only arcs of $G[U]$ of the form $u_i u_j$, $i < j$. For each ordering we perform the following polynomial-time dynamic programming procedure.

For each vertex $x \in G - U$, we define a vector $(x_0, \ldots, x_{t+1})$. Initially, set x_i to be the number of vertices $u_j \in U$ with an arc from u_j to x if $j \leq i$, or an arc from x to u_j if

$i < j$. Note that x_i is the number of arcs between x and U in the acyclic subgraph under the assumption that in the ordering of the vertices of G, x is between u_i and u_{i+1}.

Given $v, w \in V(G-U)$ and an ordering of $U \cup \{v, w\}$, an arc vw is *satisfiable* if there is no u_p such that v is after u_p and w is before u_p, for some $p \in [|U|]$. Let T be a set of arcs and let $V(T)$ be the set of end-vertices of T. For an ordering of $U \cup V(T)$, T is *satisfiable* if each arc is satisfiable, and the set T induces an acyclic subgraph.

If $G-U$ contains a block S that is itself a connected component, consider S and arbitrarily select a vertex x of S. Otherwise, find a block S in $G-U$ with only one vertex x adjacent to other vertices in $G-U$ (such a block exists as every block including an end-vertex of a longest path in $\mathrm{UN}(G)-U$ is such a block). Without loss of generality, assume that S has three vertices x, y, z (the case $|S| = 2$ can be considered similarly).

For each $i \in \{0, \ldots, t+1\}$, we let α_i be the maximum size of a set of satisfiable arcs between S and U under the restriction that x lies between u_i and u_{i+1}. Observe that $\alpha_i = \max_{j,h}(x_i + y_j + z_h + \beta(i,j,h))$, where $\beta(i,j,h)$ is the maximum size of a set of satisfiable arcs in $G[S]$ under the restriction that x lies between u_i and u_{i+1}, y lies between u_j and u_{j+1}, and z lies between u_h and u_{h+1}. Now delete $S \backslash \{x\}$ from G, and set $x_i = \alpha_i$ for each i.

Continue until each component of $G-U$ consists of a single vertex. Let x be such a single vertex, let G^* be the original graph G (i.e., given as input to our algorithm), and let X be the component of G^*-U containing x. By construction, x_i is the maximum number of satisfiable arcs from arcs in X and arcs between X and U in G^*, under the assumption x is between u_i and u_{i+1}. Since each vertex x represents a separate component, the maximum acyclic subgraph in G has $Q + \sum_{x \in V(G-U)}(\max_i x_i)$ arcs, where Q is the number of arcs $u_i u_j$ in $G[U]$ such that $i < j$.

Since the dynamic programming algorithm runs in time polynomial in n, running the algorithm for each permutation of U gives a runtime of $O((3k)!n^{O(1)})$. ◀

6 Polynomial Kernel

▶ **Lemma 12.** *Let T be a directed 3-cycle, with vertices labeled 0 or 1. Then there exists an acyclic subgraph of T with two arcs, such that there is no arc from a vertex labeled 1 to a vertex labeled 0.*

Proof. Let $V(T) = \{a, b, c\}$ and assume that a, b are labeled 0. Since T is a cycle, either the arc ac or bc exists. This arc, together with the arc between a and b, form the required acyclic subgraph. A similar argument holds when two vertices in T are labeled 1. ◀

Recall that U was introduced in Lemma 10 as the set of vertices removed by Rules 3, 4, and 5. We say that a set $\{u, a, b\}$ of vertices is a *dangerous triangle* if $u \in U$, $G[a, b]$ is a block in $G-U$, and $G[u, a, b]$ is a directed 3-cycle.

▶ **Lemma 13.** *For a vertex $u \in U$, let t_u denote the number of neighbors of u in $G-U$ which do not appear in a dangerous triangle containing u. If $t_u \geq 4k$, then we have a* YES*-instance.*

Proof. Let S denote the subgraph of $G-U$ consisting of all components C of $G-U$ which have a neighbor of u. For each component C of S, let $t_u(C)$ denote the number of neighbors of u in C which do not appear in a dangerous triangle containing u.

For each vertex $x \in G-U$, label it 0 if there exists an arc from x to u, or 1 if there is an arc from u to x. Recall from Lemma 10 each connected component in $G-U$ has at

most one block $X = \{x, y\}$ with $|X| = 2$. If one vertex x is labeled, assign y the same label. Finally, assign label 1 to any remaining unlabeled verticies in $G - U$.

We will now construct an acyclic subgraph H' of $G - U$ such that there is no arc from a vertex labeled 1 to a vertex labeled 0. We then extend this to an acyclic subgraph H containing all the arcs between u and S.

Consider each block X in $G-U$. If $|X| = 3$, and X is a directed 3-cycle, then by Lemma 12 there is an acyclic subgraph of X with two arcs. Add this to H'. Now suppose $|X| = 2$, and let a, b be the vertices of X with an arc from a to b. If $G[X \cup \{u\}]$ is a dangerous triangle, then a is labeled 1 and b is labeled 0. In this case we do not include the arc ab in H'. However, H will include the two arcs between X and u, which do not count towards $t_u(C)$. If $G[X \cup \{u\}]$ is not a dangerous triangle, then we include the arc ab in the acyclic subgraph H'. Finally, let H be the acyclic subgraph formed by adding all arcs between u and S to H'.

Observe that for each component C of S, if $G[C \cup \{u\}]$ contains no dangerous triangle then H contains at least $\gamma(C)$ arcs in $G[C]$ (by the construction of H') and $t_u(C)$ arcs between C and u (since all arcs between S and u are in H), and $\gamma(C \cup \{u\}) := \gamma(G[C \cup \{u\}]) = \gamma(C) + \frac{t_u(C)}{2} + \frac{1}{4}$. So H contains at least $\gamma(C \cup \{u\}) + \frac{t_u(C)}{2} - \frac{1}{4}$ arcs. Since $G[C \cup \{u\}]$ contains no dangerous triangle but C is adjacent to u, $t_u(C) \geq 1$, and so H contains at least $\gamma(C \cup \{u\}) + \frac{t_u(C)}{4}$ arcs.

If $G[C \cup \{u\}]$ contains a dangerous triangle then H contains at least $\gamma(C) - \frac{3}{4}$ arcs in $G[C]$ (this can be seen by contracting the arc in C appearing in the dangerous triangle, and observing that in the resulting component C', H has at least $\gamma(C')$ arcs) and $t_u(C) + 2$ arcs between C and u, and $\gamma(C \cup \{u\}) = \gamma(C) + \frac{t_u(C)+2}{2} + \frac{1}{4}$. Thus, H contains at least $\gamma(C \cup \{u\}) + \frac{t_u(C)}{2}$ arcs.

Let $C_1, C_2, \ldots, C_q$ be the components of S. Observe that $\gamma(S \cup \{u\}) = \sum_{i=1}^{q} \gamma(C_i \cup \{u\})$. Then by combining the acyclic subgraphs for each $G[C_i \cup \{u\}]$, we have that $a(G[S \cup \{u\}]) \geq \sum_{i=1}^{q} (\gamma(C_i \cup \{u\}) + \frac{t_u(C_i)}{4}) = \gamma(S \cup \{u\}) + \frac{t_u}{4}$.

Finally, observe $G - S - u$ has at most $3k$ component, since each component must contain a vertex of U. By repeated application of Lemma 2, this implies there is an acyclic subgraph of G with at least $\gamma(G) + \frac{t_u - 3k}{4}$ arcs. Hence, if $t_u \geq 4k$, we have a YES-instance. ◀

Using the above lemma and the fact that $|U| \leq 3k$ (by Lemma 10), we have that unless (G, k) is a YES-instance, there are at most $12k^2$ vertices in $G - U$ that are adjacent to a vertex in U and do not appear in a dangerous triangle with that vertex.

▶ **Lemma 14.** *Let s be the number of components in $G - U$ in which every neighbor x of a vertex $u \in U$ appears in a dangerous triangle together with u. If $s \geq k$, we have a* YES*-instance.*

Proof. By Lemma 10 such a component C_i contains at most one block of size 2. Since only blocks of size 2 can have vertices in dangerous triangles, only the vertices from this block in C_i may be adjacent to a vertex in U. But since G is reduced by Rule 1, component C_i must consist of only this block. Moreover, this block must appear in at least two dangerous triangles. Let a_i, b_i be the vertices of C_i, $i = 1, \ldots, s$ and let $C = \cup_{i=1}^{s} \{a_i, b_i\}$. Let $a_i b_i$ be an arc for each $i = 1, \ldots, s$ and note that every arc of G containing a_i (b_i, respectively) is either $a_i b_i$ or is from U to a_i (from b_i to U, respectively). Let δ_i be the number of dangerous triangles containing a_i and b_i; note that $\delta_i \geq 2$.

By (1), $G - C$ has an acyclic subgraph H with at least $\gamma(G - C)$ arcs. Observe that we can add to H all arcs entering each a_i and leaving each b_i, $i = 1, \ldots, s$, and obtain an acyclic subgraph H^* of G. We will prove that H^* contains enough arcs to show that (G, k)

is a YES-instance. Observe that $G-C$ has at most $|U| \leq 3k$ components and $G[C]$ has $2s$ vertices and $2\sum_{i=1}^{s}\delta_i + s$ arcs, and recall that each $\delta_i \geq 2$. Thus, the number of arcs in H^* is at least

$$\begin{aligned}\gamma(G-C)+2\sum_{i=1}^{s}\delta_i &\geq \frac{m-2\sum_{i=1}^{s}\delta_i - s}{2} + \frac{n-2s-3k}{4} + 2\sum_{i=1}^{s}\delta_i \\ &\geq \gamma(G) + \sum_{i=1}^{s}\delta_i - s - \frac{3k}{4} \geq \gamma(G) + \frac{k}{4}.\end{aligned}$$

◀

Let H be an undirected forest of cliques, where each block contains at most three vertices. A block B of H is called a *leaf-block* if there is at most one vertex of B belonging to another block of H. We denote the set of leaf-blocks of H by $\mathcal{L}(H)$. A block B of H is called a *path-block* if there is another block B' of H such that B and B' have a common vertex c which belongs only to these two blocks, at most one vertex of B belongs to a block other than B', and at most one vertex of B' belongs to a block other than B. We denote the set of path-blocks which are not leaf-blocks by $\mathcal{P}(H)$.

▶ **Lemma 15.** *For a forest of cliques H, with each block of size at most three, if $l = |\mathcal{L}(H)|$ and $p = |\mathcal{P}(H)|$ then $|V(H)| \leq 8l + 2p$.*

Proof. We prove the claim by induction on the number of blocks in H. The case when H has only one block is trivial. Thus, we may assume that H has at least two blocks and H is connected. Let B be a leaf-block of H, and obtain subgraph H' by deleting the vertices of B not belonging to another block. Note that $|V(H)| \leq |V(H')| + 2$.

Assume that H' has a leaf-block B' which is not a leaf-block in H. Observe that $B' \in \mathcal{P}(H)$ and by induction $|V(H)| \leq 2 + 8l + 2(p-1) \leq 8l + 2p$.

Now assume that $|\mathcal{L}(H')| = l - 1$. Observe that removal of B from H may lead to a neighbour of B, B', becoming a path-block in H', together with at most two blocks neighbouring B'. Thus, at most three blocks may become path-blocks in H'. By the induction hypothesis, $|V(H')| \leq 8(l-1)+2(p+3)$. Hence, $|V(H)| \leq 8(l-1)+2(p+3)+2 \leq 8l + 2p$. ◀

▶ **Theorem 16.** ACYCLIC SUBGRAPH ABOVE POLJAK-TURZÍK BOUND (ASAPT) *has a kernel with $O(k^2)$ vertices and $O(k^2)$ arcs.*

Proof. Consider an instance of (G^*, k) of ASAPT. Apply Rules 1 and 2 to obtain an instance (G, k) reduced by Rules 1 and 2.

Assume that (G, k) is reduced by Rules 1 and 2 and it is a NO-instance.

Now we will apply all reduction rules but Rule 2. As a result, we will obtain the set U of vertices deleted in Rules 3, 4, and 5. By Lemma 10, $|U| \leq 3k$ and, by Lemma 13, each $u \in U$ has at most $4k$ neighbors that do not appear in a dangerous triangle with u. By Lemma 14, there are at most $2k$ vertices in $G-U$ that appear in a dangerous triangle with every neighbor in U (there are at most k components, and each component has two vertices). Hence the number of neighbors in $G-U$ of vertices of U is at most $4k|U| + 2k = 12k^2 + 2k$.

Now we will adopt the terminology and notation of Lemma 15 (we extend it from UN$(G-U)$ to $G-U$ as we have done earlier). Consider a leaf-block B. Since G is reduced by Rules 1 and 3, B must contain a vertex v adjacent to U, and furthermore, v is not contained in any other block. Hence, $|\mathcal{L}(G-U)| \leq 12k^2 + 2k$.

Next, we observe that Rule 2 implies there do not exist two adjacent 3-vertex blocks $B = \{a, b, c\}$, $B' = \{c, d, e\}$ such that only a and e belong to other blocks, unless one of b, c, d has a neighbor in U. Observe that each connected component of $G - U$ contains at most one 2-vertex block, so there are at most $12k^2 + 2k$ 2-vertex path blocks. Each 2-vertex path block is adjacent to at most two 3-vertex path blocks. Hence, $|\mathcal{P}(G - U)| \leq 6(12k^2 + 2k)$. So, by Lemma 15, $|V(G - U)| \leq 8(12k^2 + 2k) + 2 \cdot 6(12k^2 + 2k) = O(k^2)$, and so $|V(G)| \leq O(k^2) + 3k = O(k^2)$.

Finally, we show G has $O(k^2)$ arcs. There are at most $|U|^2$ arcs in U. Between $G - U$ and U there are at most $(4k + 2k)|U|$ arcs. Finally, observe that $G - U$ has at most $|V(G-U)| \leq 20(12k^2+2k)$ blocks, and each block contains at most 3 arcs. Hence, $|A(G)| \leq |U|^2 + 60(12k^2 + 2k) \leq 9k^2 + 60(12k^2 + 2k) = O(k^2)$.

Thus, either (G, k) is a YES-instance, or (G, k) forms a kernel with $O(k^2)$ vertices and $O(k^2)$ arcs. ◀

7 Discussion

After this paper was submitted to FSTTCS 2012, we learned that Mnich *et al.* [15] combined modified approaches of [5] and [18] to prove that a number of graph problems parameterized above tight lower bounds are fixed-parameter tractable. In particular, they proved that ASAPT is fixed-parameter tractable. However, [15] did not obtain any results on polynomial kernels.

The algorithm of Theorem 11 has runtime $2^{O(k \log k)} n^{O(1)}$. It would be interesting to design an algorithm of runtime $2^{O(k)} n^{O(1)}$ or to prove that such an algorithm does not exist, subject to a certain complexity hypothesis, as in [13]. It would also be interesting to see whether ASAPT admits a kernel with $O(k)$ vertices.

References

1 N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo, Solving MAX-r-SAT above a tight lower bound. *Algorithmica* 61 (2011) 638–655.

2 J. Bang-Jensen and G. Gutin, Digraphs: Theory, Algorithms and Applications, Springer-Verlag, London, 2nd Ed., 2009.

3 J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon, A fixed-parameter algorithm for the directed feedback vertex set problem. J. ACM 55(5) (2008).

4 R. Crowston, M. Fellows, G. Gutin, M. Jones, F. Rosamond, S. Thomassé and A. Yeo. Simultaneously Satisfying Linear Equations Over $\mathbb{F}_2$: MaxLin2 and Max-r-Lin2 Parameterized Above Average. In *FSTTCS 2011*, LIPICS Vol. 13, 229–240, 2011.

5 R. Crowston, M. Jones and M. Mnich, Max-Cut Parameterized above the Edwards-Erdős Bound, In *ICALP 2012*, Lect. Notes Comput. Sci. 7391 (2012) 242–253.

6 M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. On multiway cut parameterized above lower bounds. In *IPEC 2011*, Lect. Notes Comput. Sci. 7112 (2011), 1–12.

7 R. G. Downey and M. R. Fellows, Parameterized Complexity. Springer-Verlag, 1999.

8 J. Flum and M. Grohe, Parameterized Complexity Theory, Springer-Verlag, 2006.

9 G. Gutin, L. van Iersel, M. Mnich, and A. Yeo, All ternary permutation constraint satisfaction problems parameterized above average have kernels with quadratic number of vertices. J. Comput. Syst. Sci. 78 (2012), 151–163.

10 G. Gutin, E. J. Kim, M. Mnich, and A. Yeo. Betweenness parameterized above tight lower bound. J. Comput. Syst. Sci. 76 (2010), 872–878.

11 G. Gutin, E.J. Kim, S. Szeider, A. Yeo, A probabilistic approach to problems parameterized above or below tight bounds, J. Comput. Syst. Sci. 77 (2011) 422–429.

12 G. Gutin and A. Yeo, Some Parameterized Problems on Digraphs. The Computer Journal 51 (2008) 363–371.

13 D. Lokshtanov, D. Marx and S. Saurabh, Slightly superexponential parameterized problems, In *SODA 2011*, 760–776, 2011.

14 M. Mahajan, V. Raman, S. Sikdar, Parameterizing above or below guaranteed values, J. Comput. System Sci. 75 (2) (2009) 137–153.

15 M. Mnich, G. Philip, S. Saurabh, and O. Suchý, Beyond Max-Cut: λ-Extendible Properties Parameterized Above the Poljak-Turzík Bound. In *FSTTCS 2012*, to appear.

16 N.S. Narayanaswamy, V. Raman, M.S. Ramanujan, and S. Saurabh, LP can be a cure for Parameterized Problems, In *STACS 2012*, LIPICS Vol. 14, 338–349, 2012.

17 R. Niedermeier. Invitation to Fixed-Parameter Algorithms, Oxford UP, 2006.

18 S. Poljak and D. Turzík, A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. Discrete Mathematics, 58 (1) (1986) 99–104.

19 V. Raman and S. Saurabh, Parameterized algorithms for feedback set problems and their duals in tournaments. Theor. Comput. Sci., 351 (3) (2006) 446–458.

20 J. Spencer, Optimal ranking of tournaments. Networks 1 (1971) 135–138.

Beyond Max-Cut: λ-Extendible Properties Parameterized Above the Poljak-Turzík Bound

Matthias Mnich[1], Geevarghese Philip[*2], Saket Saurabh[†3], and Ondřej Suchý[‡4]

1 **Cluster of Excellence, Saarbrücken, Germany.** m.mnich@mmci.uni-saarland.de
2 **Max-Planck-Institut für Informatik (MPII), Saarbrücken, Germany.** gphilip@mpi-inf.mpg.de
3 **The Institute of Mathematical Sciences, Chennai, India.** saket@imsc.res.in
4 **Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic.** ondrej.suchy@fit.cvut.cz

Abstract

Poljak and Turzík (Discrete Math. 1986) introduced the notion of λ-extendible properties of graphs as a generalization of the property of being bipartite. They showed that for any $0 < \lambda < 1$ and λ-extendible property Π, any connected graph G on n vertices and m edges contains a spanning subgraph $H \in \Pi$ with at least $\lambda m + \frac{1-\lambda}{2}(n-1)$ edges. The property of being bipartite is λ-extendible for $\lambda = 1/2$, and thus the Poljak-Turzík bound generalizes the well-known Edwards-Erdős bound for Max-Cut.

We define a variant, namely *strong* λ-extendibility, to which the Poljak-Turzík bound applies. For a strongly λ-extendible graph property Π, we define the parameterized Above Poljak-Turzík (Π) problem as follows: Given a connected graph G on n vertices and m edges and an integer parameter k, does there exist a spanning subgraph H of G such that $H \in \Pi$ and H has at least $\lambda m + \frac{1-\lambda}{2}(n-1) + k$ edges? The parameter is k, the surplus over the number of edges guaranteed by the Poljak-Turzík bound.

We consider properties Π for which the Above Poljak-Turzík (Π) problem is fixed-parameter tractable (FPT) on graphs which are $O(k)$ vertices away from being a graph in which each block is a clique. We show that for all such properties, Above Poljak-Turzík (Π) is FPT for all $0 < \lambda < 1$. Our results hold for properties of oriented graphs and graphs with edge labels.

Our results generalize the recent result of Crowston et al. (ICALP 2012) on Max-Cut parameterized above the Edwards-Erdős bound, and yield FPT algorithms for several graph problems parameterized above lower bounds. For instance, we get that the above-guarantee Max q-Colorable Subgraph problem is FPT. Our results also imply that the parameterized above-guarantee Oriented Max Acyclic Digraph problem is FPT, thus solving an open question of Raman and Saurabh (Theor. Comput. Sci. 2006).

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases Algorithms and data structures; fixed-parameter algorithms; bipartite graphs; above-guarantee parameterization.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.412

* Supported by the Indo-German Max Planck Center for Computer Science (IMPECS).
† Part of this work was done while visiting MPII supported by IMPECS.
‡ A major part of this work was done while with the Saarland University, supported by the DFG Cluster of Excellence MMCI and the DFG project DARE (GU 1023/1-2), while at TU Berlin, supported by the DFG project AREG (NI 369/9), and while visiting IMSc Chennai supported by IMPECS.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 412–423

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

1 Introduction

A number of interesting graph problems can be phrased as follows: Given a graph G as input, find a subgraph H of G with the largest number of edges such that H satisfies a specified property Π. Prominent among these is the Max-Cut problem, which asks for a *bipartite* subgraph with the maximum number of edges. A *cut* of a graph G is a partition of the vertex set of G into two parts, and the *size* of the cut is the number of edges which *cross the cut*; that is, those which have their end points in distinct parts of the partition.

Max-Cut	
Input:	A graph G and an integer k.
Question:	Does G have a cut of size at least k?

The Max-Cut problem is among Karp's original list of 21 NP-complete problems [15], and it has been extensively investigated from the point of view of various algorithmic paradigms. Thus, for example, Goemans and Williamson showed [13] that the problem can be approximated in polynomial-time within a multiplicative factor of roughly 0.878, and Khot et al. showed that this is the best possible assuming the Unique Games Conjecture [16].

Our focus in this work is on the *parameterized* complexity of a generalization of the Max-Cut problem. The central idea in the parameterized complexity analysis [8, 12] of NP-hard problems is to associate a *parameter* k with each input instance of size n, and then to ask whether the resulting *parameterized problem* can be solved in time $f(k) \cdot n^c$ where c is a constant and f is some computable function. Parameterized problems which can be solved within such time bounds are said to be fixed-parameter tractable (FPT).

The *standard parameterization* of the Max-Cut problem sets the parameter to be the size k of the cut being sought. This turns out to be not very interesting for the following reason: Let m be the number of edges in the input graph G. By an early result of Erdős [11], we know that every graph with m edges contains a cut of size at least $m/2$. Therefore, if $k \leq m/2$ then we can immediately answer YES. In the remaining case $k > m/2$, and there are less than $2k$ edges in the input graph. It follows from this bound on the size of the input that any algorithm—even a brute-force method—which solves the problem runs in FPT time on this instance.

The best lower bound known on the size of a largest cut for connected loop-less graphs on n vertices and m edges is $\frac{m}{2} + \frac{n-1}{4}$, as proved by Edwards [9, 10]. This is called the *Edwards-Erdős bound*, and it is the best possible in the sense that it is tight for an infinite family of graphs, for example, the class of cliques of odd order n. A more interesting parameterization of Max-Cut is, therefore, the following:

Max-Cut Above Tight Lower Bound (Max-Cut ATLB)	
Input:	A connected graph G, and an integer k.
Parameter:	k
Question:	Does G have a cut of size at least $\frac{m}{2} + \frac{n-1}{4} + k$?

In the work which introduced the notion of "above-guarantee" parameterization, Mahajan and Raman [17] showed that the problem of asking for a cut of size at least $\frac{m}{2} + k$ is FPT parameterized by k, and stated the fixed-parameter tractability of Max-Cut ATLB as an open problem. This question was resolved quite recently by Crowston et al. [6], who showed that the problem is in fact FPT.

We generalize the result of Crowston et al. by extending it to apply to a special case of

the so-called λ-*extendible properties*. Roughly stated[1], for a fixed $0 < \lambda < 1$ a graph property Π is said to be λ-extendible if: Given a graph $G = (V, E) \in \Pi$, an "extra" edge uv not in G, and *any set* F of "extra" edges each of which has one end point in $\{u, v\}$ and the other in V, there exists a graph $H \in \Pi$ which contains (i) all of G, (ii) the edge uv, and (iii) at least a λ fraction of the edges in F. The notion was introduced by Poljak and Turzík who showed [20] that for any λ-extendible property Π and edge-weighting function $c : E \to \mathbb{R}^+$, any connected graph $G = (V, E)$ contains a spanning subgraph $H = (V, F) \in \Pi$ such that $c(F) \geq \lambda \cdot c(E) + \frac{1-\lambda}{2} c(T)$. Here $c(X)$ denotes the total weight of all the edges in X, and T is the set of edges in a minimum-weight spanning tree of G. It is not difficult to see that the property of being bipartite is λ-extendible for $\lambda = 1/2$, and so—once we assign unit weights to all edges—the Poljak and Turzík result implies the Edwards-Erdős bound. Other examples of λ-extendible properties—with different values of λ—include q-colorability and acyclicity in oriented graphs.

In this work we study the natural above-guarantee parameterized problem for λ-extendible properties Π, which is: given a connected graph $G = (V, E)$ and an integer k as input, does G contain a spanning subgraph $H = (V, F) \in \Pi$ such that $c(F) = \lambda \cdot c(E) + \frac{1-\lambda}{2} c(T) + k$? To derive a generic FPT algorithm for this class of problems, we use the "reduction" rules of Crowston et al. To make these rules work, however, we need to make a couple of concessions. Firstly, we slightly modify the notion of lambda extendibility; we define a (potentially) stronger notion which we name *strong* λ-*extendibility*. Every strongly λ-extendible property is also λ-extendible by definition, and so the Poljak-Turzík bound applies to strongly λ-extendible properties as well. Observe that for each way of assigning edge-weights, the Poljak and Turzík result yields a (potentially) different lower bound on the weight of the subgraph. Following the spirit of the question posed by Mahajan and Raman and solved by Crowston et al., we choose from among these the lower bound implied by the unit-edge-weighted case. This is our second simplification, and for this "unweighted" case the Poljak and Turzík result becomes: for any strongly λ-extendible property Π, any connected graph $G = (V, E)$ contains a spanning subgraph $H = (V, F) \in \Pi$ such that $|F| = \lambda \cdot |E| + \frac{1-\lambda}{2}(|V| - 1)$.

The central problem which we discuss in this work is thus the following; here $0 < \lambda < 1$, and Π is an arbitrary—but fixed—strongly λ-extendible property:

Above Poljak-Turzík (Π) (APT(Π))	
Input:	A connected graph $G = (V, E)$ and an integer k.
Parameter:	k
Question:	Is there a spanning subgraph $H = (V, F) \in \Pi$ of G such that $\lvert F\rvert \geq \lambda\lvert E\rvert + \frac{1-\lambda}{2}(\lvert V\rvert - 1) + k$?

1.1 Our Results and their Implications

We show that the Above Poljak-Turzík (Π) problem is FPT for every strongly λ-extendible property Π for which APT(Π) is FPT on a class of "almost-forests of cliques". Informally[1], this is a class of graphs which are a small number ($O(k)$) of vertices away from being a graph in which each block is a clique. This requirement is satisfied by the properties underlying a number of interesting problems, including Max-Cut, Max q-Colorable Subgraph, and Oriented Max Acyclic Digraph. The main result of this paper is the following.

[1] See subsection 1.2 and section 2 for the definitions of various terms used in this section.

▶ **Theorem 1.** *The* Above Poljak-Turzík (Π) *problem is fixed-parameter tractable for a* λ*-extendible property* Π *of graphs if*

- Π *is strongly* λ*-extendible and*
- Above Poljak-Turzík (Π) *is FPT on almost-forests of cliques.*

This also holds for such properties of oriented and/or labelled graphs.

We prove Theorem 1 using the classical "Win/Win" approach of parameterized complexity. To wit: given an instance (G, k) of a strongly λ-extendible property Π, in polynomial time we either (i) show that (G, k) is a yes instance, or (ii) find a vertex subset S of G of size at most $6k/(1-\lambda)$ such that deleting S from G leaves a "forest of cliques". To do this we use the "reduction" rules derived by Crowston et al. [6] to solve Max-Cut. Our main technical contribution is a proof that these rules are sufficient to show that *every* NO *instance* of APT(Π) is at a vertex-deletion distance of $O(k)$ from a forest of cliques. This proof requires several new ideas: a result which holds for *all* strongly λ-extendible properties Π is a significant step forward from Max-Cut. Our main result unifies and generalizes several known FPT results, and implies new ones. These include such algorithms for (i) Max-Cut, (ii) finding a q-colorable subgraph of the maximum size, and (ii) finding a maximum-size acyclic subdigraph in an oriented graph. We also obtain a linear *vertex* kernel for the latter problem, complementing the quadratic *arc* kernel by Gutin et al. [14].

Related Work. The notion of parameterizing above (or below) some kind of "guaranteed" values—lower and upper bounds, respectively—was introduced by Mahajan and Raman [17]. It has proven to be a fertile area of research, and Max-Cut is now just one of a host of interesting problems for which we have FPT results for such questions [1, 2, 3, 4, 5, 6, 14, 18, 21]. Indeed, very recent work due to Crowston et al. [4], where they take up an above-guarantee parameterization of the problem of finding a large acyclic digraph in an oriented graph, appears as an article in these selfsame proceedings; see the discussion after Corollary 32.

1.2 Preliminaries

We use "graph" to denote simple graphs without self-loops, directions, or labels, and assume the graph terminology of Diestel [7]. $V(G)$ and $E(G)$ denote, respectively, the vertex and edge sets of a graph G. For $S \subseteq V(G)$, we use (i) $G[S]$ to denote the subgraph of G induced by the set S, (ii) $G \setminus S$ to denote $G[V(G) \setminus S]$, (iii) $\delta(S)$ to denote the set of edges in G which have exactly one end-point in S, and (iv) $e_G(S)$ to denote $|E(G[S])|$; we omit the subscript G if it is clear from the context. A *clique* in a graph G is a set of vertices C such that between any pair of vertices in C there is an edge in $E(G)$. A *block* of graph G is a maximal 2-connected subgraph of G, and a graph G is a *forest of cliques* if the vertex set of every block forms a clique. A *leaf clique* of a forest of cliques is a block of the graph which contains at most one cutvertex of the graph.

For $F \subseteq E(G)$, (i) we use $G \setminus F$ to denote the graph $(V(G), E(G) \setminus F)$, and (ii) for a weight function $c : E(G) \to \mathbb{R}^+$, we use $c(F)$ to denote the sum of the weights of all the edges in F. A *graph property* is a collection of graphs. For $i, j \in \mathbb{N}$ we use K_i to denote the complete graph on i vertices, and $K_{i,j}$ to denote the complete bipartite graph in which the two parts of vertices are of sizes i, j.

Our results also apply to graphs with oriented edges, and those with edge labels. Subgraphs of an oriented or labelled graph G inherit the orientation or labelling—as is the case—of G in the natural manner: each surviving edge keeps the same orientation/labelling as it had in G. For a graph G of any kind, we use G_S to denote the simple graph obtained by removing

all orientations and labels from G; we say that G is connected (or contains a clique, and so forth) if G_S is connected (or contains a clique, and so forth).

2 Definitions

The following notion is a variation on the concept of λ-extendibility [20].

▶ **Definition 2** (Strong λ-extendibility)**.** Let $\mathcal{G}$ be the class of (possibly oriented and/or labelled) graphs, and let $0 < \lambda < 1$. A property $\Pi \subseteq \mathcal{G}$ is *strongly λ-extendible* if it satisfies the following:

Inclusiveness $\{G \in \mathcal{G} \mid G_S \in \{K_1, K_2\}\} \subseteq \Pi$

Block additivity $G \in \mathcal{G}$ belongs to Π if and only if each block of G belongs to Π.

Strong λ-subgraph extension Let $G \in \mathcal{G}$ and $S \subseteq V(G)$ be such that $G[S] \in \Pi$ and $G \setminus S \in \Pi$. For any weight function $c : E(G) \to \mathbb{R}^+$ there exists an $F \subseteq \delta(S)$ with $c(F) \geq \lambda \cdot c(\delta(S))$, such that $G \setminus (\delta(S) \setminus F) \in \Pi$.

The strong λ-subgraph extension requirement can be rephrased as follows: Let $V(G) = X \uplus Y$ be a cut of graph G such that $G[X], G[Y] \in \Pi$, and let F be the set of edges which cross the cut. For *any* weight function $c : F \to \mathbb{R}^+$, there exists a subset $F' \subseteq F$ such that (i) $c(F') \leq (1 - \lambda) \cdot c(F)$, and (ii) $(G \setminus F') \in \Pi$. Informally, one can pick a λ-fraction of the cut and delete the rest to obtain a graph which belongs to Π.

We recover Poljak and Turzík's definition of λ-extendibility from the above definition by replacing strong λ-subgraph extension with the following property:

λ-edge extension Let $G \in \mathcal{G}$ and $S \subseteq V(G)$ be such that $G_S[S]$ *is isomorphic to* K_2 and $G \setminus S \in \Pi$. For any weight function $c : E(G) \to \mathbb{R}^+$ there exists an $F \subseteq \delta(S)$ with $c(F) \geq \lambda \cdot c(\delta(S))$, such that $G \setminus (\delta(S) \setminus F) \in \Pi$.

Observe from the definitions that any graph property which is strongly λ-extendible is also λ-extendible. It follows that Poljak and Turzík's result for λ-extendible properties applies also to strongly λ-extendible properties.

▶ **Theorem 3** (Poljak-Turzík bound)**.** [**20**] *Let $\mathcal{G}$ be a class of (possibly oriented and/or labelled) graphs. Let $0 < \lambda < 1$, and let $\Pi \subseteq \mathcal{G}$ be a strongly λ-extendible property. For any connected graph $G \in \mathcal{G}$ and weight function $c : E(G) \to \mathbb{R}^+$, there exists a spanning subgraph $H \in \Pi$ of G such that $c(E(H)) \geq \lambda \cdot c(E(G)) + \frac{1-\lambda}{2} c(T)$, where T is the set of edges in a minimum-weight spanning tree of G_S.*

When all edges are assigned weight 1, we get:

▶ **Corollary 4.** *Let $\mathcal{G}, \lambda, \Pi$ be as in Theorem 3. Any connected graph $G \in \mathcal{G}$ on n vertices and m edges has a spanning subgraph $H \in \Pi$ with at least $\lambda m + \frac{1-\lambda}{2}(n-1)$ edges.*

Our results apply to properties which satisfy the additional requirement of being FPT on almost-forests of cliques.

▶ **Definition 5** (FPT on almost-forests of cliques)**.** Let $0 < \lambda < 1$, and let Π be a strongly λ-extendible property (of graphs with or without orientations/labels). The Structured Above Poljak-Turzík (Π) problem is a variant of the Above Poljak-Turzík (Π) problem in which, along with the graph G and $k \in \mathbb{N}$, the input contains a set $S \subseteq V(G)$ such that $|S| = O(k)$ and $G \setminus S$ is a forest of cliques. We say that the property Π is *FPT on almost-forests of cliques* if the Structured Above Poljak-Turzík (Π) problem is FPT.

In other words, a λ-extendible property Π is FPT on almost-forests of cliques, if for any constant q there is an algorithm that, given a connected graph G, k and a set $S \subseteq V(G)$ of size at most $q \cdot k$ such that $G \setminus S$ is a forest of cliques, correctly decides whether (G, k) is a yes-instance of APT(Π) in $O(f(k) \cdot n^{O(1)})$ time, for some computable function f.

3 Fixed-Parameter Algorithms for Above Poljak-Turzík (Π)

We now prove Theorem 1 using Crowston et al.'s line of attack for solving Max-Cut [6]. The crux of their strategy is a polynomial-time procedure which takes the input (G, k) of Max-Cut and finds a subset $S \subseteq V(G)$ such that (i) $G \setminus S$ is a forest of cliques, and (ii) if (G, k) is a NO instance, then $|S| \leq 3k$. Thus if $|S| > 3k$, then one can immediately answer YES; otherwise one solves the problem in FPT time using the fact that Max-Cut is FPT on almost-forests of cliques (Definition 5).

The nontrivial part of our work consists of proving that the procedure for Max-Cut applies also to the much more general family of strongly λ-extendible problems, where the bound on the size of S depends on λ. To do this, we show that each of the four rules used for Max-Cut is safe to apply for any strongly λ-extendible property. From this we get

▶ **Lemma 6.** *Let $0 < \lambda < 1$, and let Π be a strongly λ-extendible graph property. Given a connected graph G with n vertices and m edges and an integer k, in polynomial time we can do one of the following:*

1. *Decide that there is a spanning subgraph $H \in \Pi$ of G with at least $\lambda m + \frac{1-\lambda}{2}(n-1) + k$ edges, or;*
2. *Find a set S of at most $\frac{6}{1-\lambda}k$ vertices in G such that $G \setminus S$ is a forest of cliques.*

This also holds for strongly λ-extendible properties of oriented and/or labelled graphs.

We give an algorithmic proof of Lemma 6. Let (G, k) be an instance of Above Poljak-Turzík (Π). The algorithm initially sets $\tilde{G} := G$, $\tilde{S} := \emptyset$, $\tilde{k} := k$, and then applies a series of rules to the tuple $(\tilde{G}, \tilde{S}, \tilde{k})$. Each application of a rule to $(\tilde{G}, \tilde{S}, \tilde{k})$ produces a tuple (G', S', k') such that (i) if $\tilde{G} \setminus \tilde{S}$ is connected then so is $G' \setminus S'$, and (ii) if $(\tilde{G} \setminus \tilde{S}, \tilde{k})$ is a NO instance of APT(Π) then so is $(G' \setminus S', k')$; the converse may not hold. The algorithm then sets $\tilde{G} := G', \tilde{S} := S', \tilde{k} := k'$, and repeats the process, till none of the rules applies.

We now state the four rules—which, but for minor changes, are due to Crowston et al. [6]—and show that they suffice to prove Lemma 6. We assume throughout that λ and Π are as in Lemma 6. For brevity we assume that the empty graph is in Π, and we let $\lambda' = \frac{1}{2}(1-\lambda)$ so that $\lambda + 2\lambda' = 1$.

▶ **Rule 1.** Let $\tilde{G} \setminus \tilde{S}$ be connected. If $v \in (V(\tilde{G}) \setminus \tilde{S})$ and $X \subseteq (V(\tilde{G}) \setminus (\tilde{S} \cup \{v\}))$ are such that (i) $\tilde{G}[X]$ is a connected component of $\tilde{G} \setminus (\tilde{S} \cup \{v\})$, and (ii) $X \cup \{v\}$ is a clique in $\tilde{G}$, then delete X from $\tilde{G}$ to get G'; set $S' = \tilde{S}, k' = \tilde{k}$.

▶ **Rule 2.** Let $\tilde{G} \setminus \tilde{S}$ be connected. Suppose Rule 1 does not apply, and let $X_1, \ldots, X_\ell$ be the connected components of $\tilde{G} \setminus (\tilde{S} \cup \{v\})$ for some $v \in V(\tilde{G}) \setminus \tilde{S}$. If at least one of the X_is is a clique, and at most one of them is *not* a clique, then

- Delete all the X_is which are cliques—let these be d in number—to get G', and
- Set $S' := \tilde{S} \cup \{v\}$ and $k' := \tilde{k} - d\lambda'$.

▶ **Rule 3.** Let $\tilde{G} \setminus \tilde{S}$ be connected. If $a, b, c \in V(\tilde{G}) \setminus \tilde{S}$ are such that $\{a, b\}, \{b, c\} \in E(\tilde{G})$, $\{a, c\} \notin E(\tilde{G})$, and $\tilde{G} \setminus (\tilde{S} \cup \{a, b, c\})$ is connected, then

- Set $S' := \tilde{S} \cup \{a, b, c\}$ and $k' := \tilde{k} - \lambda'$.

▶ **Rule 4.** Let $\tilde{G} \setminus \tilde{S}$ be connected. Suppose Rule 3 does not apply, and let $x, y \in V(\tilde{G}) \setminus \tilde{S}$ be such that $\{x, y\} \notin E(\tilde{G})$. Let $C_1, \ldots, C_\ell$ be the connected components of $\tilde{G} \setminus (\tilde{S} \cup \{x, y\})$. If there is at least one C_i such that (i) both $V(C_i) \cup \{x\}$ and $V(C_i) \cup \{y\}$ are cliques in $\tilde{G} \setminus \tilde{S}$, and (ii) there is at most one C_i for which (i) does *not* hold, then

- Delete all the C_is which satisfy condition (i) above to get G', and,
- Set $S' := \tilde{S} \cup \{x, y\}$, $k' := \tilde{k} - \lambda'$.

Let $(G^\star, S, k^\star)$ be the tuple which we get by applying these rules exhaustively to the input tuple $(G, \emptyset, k)$. To prove Lemma 6, it is sufficient to prove the following claims: (i) the rules can be exhaustively applied in polynomial time; (ii) $G \setminus S$ is a forest of cliques; (iii) the rules transform NO-instances to NO-instances; and, (iv) if (G, k) is a NO instance, then $|S| \leq \frac{6}{1-\lambda} k$. We now proceed to prove these over several lemmata. Our rules are identical to those of Crowston et al. in how the rules modify the graph; the only difference is in how we change the parameter k. The first two claims thus follow directly from their work.

▶ **Lemma 7.** [⋆][2] *Rules 1 to 4 can be exhaustively applied to an instance (G, k) of* Above Poljak-Turzík (Π) *in polynomial time. The resulting tuple $(G^\star, S, k^\star)$ has $|V(G^\star) \setminus S| \leq 1$.*

▶ **Lemma 8.** **[6, Lemma 8]** *Let $(G^\star, S, k^\star)$ be the tuple obtained by applying Rules 1 to 4 exhaustively to an instance (G, k) of* Above Poljak-Turzík (Π)*. Then $G \setminus S$ is a forest of cliques.*

The correctness of the remaining two claims is a consequence of the λ-extendibility of property Π, and we make critical use of this fact in building the rest of our proof. This is the one place where this work is significantly different from the work of Crowston et al.; they could take advantage of the special characteristics of *one specific* property, namely bipartitedness, to prove the analogous claims for Max-Cut.

We say that a rule is *safe* if it preserves NO instances.

▶ **Definition 9.** Let $(\tilde{G}, \tilde{S}, \tilde{k})$ be an arbitrary tuple to which one of the rules 1, 2, 3, or 4 applies, and let (G', S', k') be the resulting tuple. We say that the rule is *safe* if, whenever $(G' \setminus S', k')$ is a YES instance of Above Poljak-Turzík (Π), then so is $(\tilde{G} \setminus \tilde{S}, \tilde{k})$.

We now prove that each of the four rules is safe. For a graph G we use $val(G)$ to denote the maximum number of edges in a subgraph $H \in \Pi$ of G, and $pt(G)$ to denote the Poljak-Turzík bound for G. Thus if G is connected and has n vertices and m edges then $pt(G) = \lambda m + \lambda'(n-1)$, and Corollary 4 can be written as $val(G) \geq pt(G)$. For each rule we assume that $G' \setminus S'$ has a spanning subgraph $H' \in \Pi$ with at least $pt(G' \setminus S') + k'$ edges, and show that $\tilde{G} \setminus \tilde{S}$ has a spanning subgraph $\tilde{H} \in \Pi$ with at least $pt(\tilde{G} \setminus \tilde{S}) + \tilde{k}$ edges.

We first derive a couple of lemmas which describe how contributions from subgraphs of a graph G add up to yield lower bounds on $val(G)$.

▶ **Lemma 10.** [⋆] *Let v be a cutvertex of a connected graph G, and let $\mathcal{C} = C_1, C_2, \ldots C_r; r \geq 2$ be sets of vertices of G such that (i) for every $i \neq j$ we have $C_i \cap C_j = \{v\}$, (ii) there is no edge between $C_i \setminus \{v\}$ and $C_j \setminus \{v\}$, and (iii) $\bigcup_{1 \leq i \leq r} C_i = V(G)$. For $1 \leq i \leq r$, let $H_i \in \Pi$ be a subgraph of $G[C_i]$ with $pt(G[C_i]) + k_i$ edges, and let $H = (V(G), \bigcup_{i=1}^{r} E(H_i))$. Then H is a subgraph of G, $H \in \Pi$, and $|E(H)| \geq pt(G) + \sum_{i=1}^{r} k_i$.*

[2] Proofs of results marked with a ⋆ have been deferred to the full version of the paper, a preprint of which is available on arXiv [19].

▶ **Lemma 11.** [⋆] *Let G be a graph, and let $S \subseteq V(G)$ be such that there exist (i) a subgraph $H_S \in \Pi$ of $G[S]$ with at least $pt(G[S]) + \lambda' + k_S$ edges, and (ii) a subgraph $\overline{H} \in \Pi$ of $G \setminus S$ with at least $pt(G \setminus S) + \lambda' + \overline{k}$ edges. Then there is a subgraph $H \in \Pi$ of G with at least $pt(G) + \lambda' + k_S + \overline{k}$ edges.*

This lemma has a useful special case which we state as a corollary:

▶ **Corollary 12.** *Let G be a graph, and let $S \subseteq V(G)$ be such that (i) there exists a subgraph $H_S \in \Pi$ of $G[S]$ with at least $pt(G[S]) + \lambda' + k_S$ edges, and (ii) the subgraph $G \setminus S$ has a perfect matching. Then there is a subgraph $H \in \Pi$ of G with at least $pt(G) + \lambda' + k_S$ edges.*

Proof. Recall that the graph K_2 is in Π by definition, and observe that $pt(K_2) = \lambda + \lambda'$. Thus K_2 has $pt(K_2) + \lambda'$ edges. The corollary now follows by repeated application of Lemma 11, each time considering a new edge of the matching as the graph $\overline{H}$. ◀

The safeness of Rule 1 is now a consequence of the block additivity property.

▶ **Lemma 13.** [⋆] *Rule 1 is safe.*

We now prove some useful facts about certain simple graphs, in the context of strongly λ-extendible properties. Observe that every block of a forest is one of $\{K_1, K_2\}$, which are both in Π. From this and the block additivity property of Π we get

▶ **Observation 14.** Every forest (with every orientation and labeling) is in Π.

The graph $K_{2,1}$ is a useful special case.

▶ **Observation 15.** The graph $K_{2,1}$—also with any kind of orientation or labelling—is in Π, and it has $pt(K_{2,1}) + \lambda' + \lambda'$ edges.

The graph obtained by removing one edge from K_4 is another useful object, since it always has more edges than its Poljak-Turzík bound.

▶ **Lemma 16.** [⋆] *Let G be a graph formed from the graph K_4—also with any kind of orientation or labelling—by removing one edge. Then (i) $val(G) \geq 3$, (ii) $val(G) \geq 4$ if $\lambda > 1/3$, and (iii) $val(G) = 5$ if $\lambda > 1/2$. As a consequence,*

$$val(G) \geq pt(G) + \lambda' + \begin{cases} (1 - 3\lambda) & \text{if } \lambda \leq 1/3, \\ (2 - 3\lambda) & \text{if } 1/3 < \lambda \leq 1/2, \text{ and,} \\ (3 - 3\lambda) & \text{if } \lambda > 1/2. \end{cases} \tag{1}$$

The above lemmata help us prove that Rules 2 and 3 are safe.

▶ **Lemma 17.** [⋆] *Rule 2 is safe.*

Following the notation of Rule 3, observe that for the vertex subset $T = \{a, b, c\} \subseteq V(\tilde{G} \setminus \tilde{S})$ we have—from Observation 15—that $\tilde{G}[T] \in \Pi$ and $val(T) \geq pt(T) + \lambda' + \lambda'$. Since $G' \setminus S' = (\tilde{G} \setminus \tilde{S}) \setminus T$, if $val(G' \setminus S') \geq pt(G' \setminus S') + k'$ then applying Lemma 11 we get that $val(\tilde{G} \setminus \tilde{S}) \geq pt(\tilde{G} \setminus \tilde{S}) + \lambda' + k' = pt(\tilde{G} \setminus \tilde{S}) + \tilde{k}$. Hence we get

▶ **Lemma 18.** *Rule 3 is safe.*

To show that Rule 4 is safe, we need a number of preliminary results. We first observe that—while the rule is stated in a general form—the rule only ever applies when it can delete exactly one component.

▶ **Observation 19.** [⋆] Whenever Rule 4 applies, there is exactly one component to be deleted, and this component has at least 2 vertices.

Our next few lemmas help us further restrict the structure of the subgraph to which Rule 4 applies. We start with a result culled from Crowston et al.'s analysis of the four rules.

▶ **Lemma 20.** [6][⋆] *If none of Rules 1, 2, and 3 applies to* $(\tilde{G}, \tilde{S}, \tilde{k})$*, and Rule 4* does *apply, then one can find*

- *A vertex* $r \in V(\tilde{G} \setminus \tilde{S})$ *and a set* $X \subseteq V(\tilde{G} \setminus \tilde{S})$ *such that* X *is a connected component of* $\tilde{G} \setminus (\tilde{S} \cup \{r\})$*, and the graph* $(\tilde{G} \setminus \tilde{S})[X \cup \{r\}]$ *is 2-connected;*
- *Vertices* $x, y \in X$ *such that* $\{x, y\} \notin E(\tilde{G})$ *and*
  - $(\tilde{G} \setminus \tilde{S}) \setminus \{x, y\}$ *has exactly two components* G', C*,*
  - $r \in G'$*;* $C \cup \{x\}, C \cup \{y\}$ *are cliques, and each of* x, y *is adjacent to some vertex in* G'

From this we get the following.

▶ **Lemma 21.** [⋆] *Suppose Rules 1, 2, and 3 do not apply, and Rule 4 applies. Then we can apply Rule 4 in such a way that if* x, y *are the vertices to be added to* $\tilde{S}$ *and* C *the clique to be deleted, then* $N(x) \cup N(y) \setminus (C \cup \tilde{S})$ *contains at most one vertex* z *such that* $\tilde{G} \setminus (\tilde{S} \cup \{z\})$ *is disconnected.*

We now show that in such a case $N(x) \setminus (C \cup \tilde{S}) = N(y) \setminus (C \cup \tilde{S}) = \{r\}$, and so the graph $\tilde{G} \setminus (\tilde{S} \cup \{r\})$ is not connected. First we need the following simple lemma.

▶ **Lemma 22.** [⋆] *Whenever Rule 4 applies, with* x, y *the vertices to be added to* $\tilde{S}$ *and* C *the clique to be deleted, every* u *in* $N(x) \setminus (C \cup \tilde{S})$ *is a cutvertex in* $\tilde{G} \setminus (\tilde{S} \cup \{x\})$ *and every* u *in* $N(y) \setminus (C \cup \tilde{S})$ *is a cutvertex in* $\tilde{G} \setminus (\tilde{S} \cup \{y\})$*.*

This allows us to enforce a very special way of applying Rule 4.

▶ **Lemma 23.** [⋆] *Suppose Rules 1, 2, and 3 do not apply, and Rule 4 applies. Then we can apply Rule 4 in such a way that if* x, y *are the vertices to be added to* $\tilde{S}$ *and* C *the clique to be deleted, then* $N(x) \setminus (C \cup \tilde{S}) = N(y) \setminus (C \cup \tilde{S}) = \{z\}$*, and* $\tilde{G} \setminus (\tilde{S} \cup \{z\})$ *is disconnected.*

These lemmas help us prove that Rule 4 is safe.

▶ **Lemma 24.** [⋆] *Rule 4 is safe.*

The next lemma gives us a bound on the size of the set S which we compute.

▶ **Lemma 25.** [⋆] *Let* $\tilde{G}$ *be a connected graph,* $\tilde{S} \subseteq V(\tilde{G})$*, and* $\tilde{k} \in \mathbb{N}$*, and let one application of Rule 1, 2, 3, or 4 to* $(\tilde{G}, \tilde{S}, \tilde{k})$ *result in the tuple* (G', S', k')*. Then* $|S' \setminus \tilde{S}| \leq 3(\tilde{k} - k')/\lambda'$*.*

Now we are ready to prove Lemma 6, from which our main result (Theorem 1) easily follows.

Proof (of Lemma 6). Let (G, k) be an input instance of Above Poljak-Turzík (Π), and let $(G^\star, S, k^\star)$ be the tuple which we get by applying the four rules exhaustively to the tuple $(G, \emptyset, k)$. From Lemma 7 we know that this can be done in polynomial time, and that the resulting graph satisfies $|V(G^\star) \setminus S| \leq 1$.

Thus $G^\star \setminus S$ is either K_1 or the empty graph, and so $G^\star \setminus S \in \Pi$ and $pt(G^\star \setminus S) = 0, |E(G^\star \setminus S)| = 0$. Hence if $k^\star \leq 0$ then $(G^\star \setminus S, k^\star)$ is a YES instance of Above Poljak-Turzík (Π). Since all the four rules are safe—Lemmas 13, 17, 18, and 24—we get that in this case (G, k) is a YES instance, and we can return YES. On the other hand if $k^\star > 0$ then we know—using Lemma 25—that $|S| < 3k/\lambda' = 6k/(1 - \lambda)$, and—from Lemma 8—that $G \setminus S$ is a forest of cliques. This completes the proof. ◀

4 Applications

In this section we use Theorem 1 to show that Above Poljak-Turzík (II) is FPT for almost all natural examples of λ-extendible properties listed by Poljak and Turzík [20]. For want of space, we defer the definitions and all proofs to the full version of the paper.

4.1 Application to Partitioning Problems

First we focus on properties specified by a homomorphism to a vertex transitive graph. As a graph is h-colorable if and only if it has a homomorphism to K_h, searching for a maximal h-colorable subgraph is one of the problems resolved in this section. In particular, a maximum cut equals a maximum bipartite subgraph and, hence, is also one of the properties studied in this section. We use $\mathcal{G}$ to denote the class of graphs—oriented or edge-labelled—to which the property in question belongs.

It is not difficult to see that every vertex-transitive graph G is a regular graph. In particular, if $\mathcal{G}$ allows labels and/or orientations, then for every label l and every orientation r each vertex of a vertex transitive graph G is incident to the same number of edges of label l and orientation r. Let us denote this number $d_{l,r}(G)$. Let us also denote by $d_{\mathcal{G}}(G)$ the minimum of $d_{l,r}(G)$ over all labels and orientations allowed in $\mathcal{G}$.

▶ **Lemma 26.** [⋆] *Let G_0 be a vertex-transitive graph with at least one edge of every label and orientation allowed in $\mathcal{G}$. Then the property "to have a homomorphism to G_0" is strongly d/n_0-extendible in $\mathcal{G}$, where n_0 is the number of vertices of G_0 and $d = d_{\mathcal{G}}(G_0)$.*

Note that while the above lemma poses no restrictions on the graphs considered, we can prove the following only for simple graphs.

▶ **Lemma 27.** [⋆] *If G_0 is an unoriented unlabeled graph, then the problem APT("to have a homomorphism into G_0") is FPT on almost-forests of cliques.*

Lemma 26 and Lemma 27 together with Theorem 1 imply the following corollary.

▶ **Corollary 28.** *The problem APT("to have a homomorphism into G_0") is fixed-parameter tractable for every unoriented unlabeled vertex transitive graph G_0.*

In particular, by setting $G_0 = K_q$ we get the following result.

▶ **Corollary 29.** *Given a graph G with m edges and n vertices and an integer k, it is FPT to decide whether G has an q-colorable subgraph with at least $m \cdot (q-1)/q + (n-1)/(2q) + k$ edges.*

This shows that the Max q-Colorable Subgraph problem is FPT when parameterized above the Poljak and Turzík bound [20].

4.2 Finding Acyclic Subgraphs of Oriented Graphs

In this section we show how to apply our result to the problem of finding a maximum-size directed acyclic subgraph of an oriented graph, where the size of the subgraph is defined as the number of arcs in the subgraph. Recall that an oriented graph is a directed graph where between any two vertices there is at most one arc. We show that Theorem 1 applies to this problem. To this end we need the following two lemmata.

▶ **Lemma 30.** [⋆] *The property "acyclic oriented graphs" is strongly $1/2$-extendible in the class of oriented graphs.*

▶ **Lemma 31.** [⋆] *The problem APT("acyclic oriented graphs") is FPT on almost-forests of cliques.*

Combining Lemmata 30 and 31 with Theorem 1 we get the following corollary.

▶ **Corollary 32.** *The problem APT("acyclic oriented graphs") is fixed-parameter tractable.*

To put this result in some context, we recall a couple of open problems posed by Raman and Saurabh [21]: Are the following questions FPT parameterized by k?

- Given an oriented directed graph on n vertices and m arcs, does it have a subset of at least $\frac{m}{2} + \frac{1}{2}(\lceil \frac{n-1}{2} \rceil) + k$ arcs that induces an acyclic subgraph?
- Given a directed graph on n vertices and m arcs, does it have a subset of at least $m/2 + k$ arcs that induces an acyclic subgraph?

In the first question, a "more correct" lower bound is the one of Poljak and Turzík, i.e., $\frac{m}{2} + \frac{1}{2}\frac{(n-1)}{2}$, and the lower bound is true only for connected graphs. Without the connectivity requirement, the problem is NP-hard already for $k = 0$: The reduction is from the NP-hard problem of finding if a connected oriented graph has an acyclic subgraph with $\frac{m}{2} + \frac{1}{2}\frac{(n-1)}{2} + k$ arcs, and consists of adding $4k$ disjoint oriented 3-cycles. Corollary 32 answers the corrected question. Crowston et al. take up this problem in their recent independent work [4], a version of which appears as another article in these proceedings. In addition to showing that the problem is FPT, they also derive an $O(k^2)$ kernel for the problem.

For the second question, observe that each maximal acyclic subgraph contains exactly one arc from every pair of opposite arcs. Hence we can remove these pairs from the digraph without changing the relative solution size, as exactly half of the removed arcs can be added to any solution to the modified instance. Thus, we can restrict ourselves to oriented graphs.

Now suppose that the oriented graph which we have as input is disconnected. It is easy to check that picking two vertices from different connected components and identifying them does not change the solution size, as this way we never create a cycle from an acyclic graph. After applying this reduction rule exhaustively, the digraph becomes an oriented connected graph, and the parameter is unchanged. But then if $k \leq (n-1)/4$ then $m/2 + k \leq m/2 + (n-1)/4$ and we can answer YES due to Corollary 4. Otherwise $n \leq 4k$, we have a linear vertex kernel, and we can solve the problem by the well known dynamic programming on the kernel [22]. The total running time of this algorithm is $O(2^{4k} \cdot k^2 + m)$. The smallest kernel previously known for this problem is by Gutin et al., and has a quadratic number of arcs [14].

5 Conclusion and Open Problems

In this paper we studied a generalization of the graph property of being bipartite, from the point of view of parameterized algorithms. We showed that for every strongly λ-extendible property Π which satisfies an additional "solvability" constraint, the Above Poljak-Turzík (Π) problem is FPT. As an illustration of the usefulness of this result, we obtained FPT algorithms for the above-guarantee versions of three graph problems.

Note that for each of the three problems—Max-Cut, Max q-Colorable Subgraph, and Oriented Max Acyclic Digraph—for which we used Theorem 1 to derive FPT algorithms for the above-guarantee question, we needed to devise a separate FPT algorithm which works for graphs that are at a vertex deletion distance of $O(k)$ from forests of cliques. We leave open the important question of finding a right logic that captures these examples, and of showing that any problem expressible in this logic is FPT parameterized by deletion distance to forests of cliques. We also leave open the kernelization complexity question for λ-extendible properties.

References

1 N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving Max-r-Sat above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011.

2 B. Bollobas and A. Scott. Better bounds for max cut. *Contemporary Combinatorics, Bolyayi Society Mathematical Studies*, 10:185–246, 2002.

3 R. Crowston, M. R. Fellows, G. Gutin, M. Jones, F. A. Rosamond, S. Thomassé, and A. Yeo. Simultaneously satisfying linear equations over $\mathbb{F}_2$: MaxLin2 and Max-r-Lin2 parameterized above average. In *FSTTCS 2011*, volume 13 of *LIPIcs*, pages 229–240.

4 R. Crowston, G. Gutin, and M. Jones. Directed Acyclic Subgraph Problem Parameterized above the Poljak-Turzik Bound. *CoRR*, abs/1207.3586, 2012.

5 R. Crowston, G. Gutin, M. Jones, V. Raman, and S. Saurabh. Parameterized complexity of MaxSat above average. In *LATIN 2012*, volume 7256 of *LNCS*, pages 184–194, 2012.

6 R. Crowston, M. Jones, and M. Mnich. Max-cut parameterized above the Edwards-Erdős bound. In *ICALP 2012*, volume 7391 of *LNCS*, pages 242–253. 2012.

7 R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2010.

8 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

9 C. S. Edwards. Some extremal properties of bipartite subgraphs. *Canadian Journal of Mathematics*, 25:475–483, 1973.

10 C. S. Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Recent Advances in Graph Theory*, pages 167–181. 1975.

11 P. Erdős. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3:113–116, 1965.

12 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

13 M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:6, 1995.

14 G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. A probabilistic approach to problems parameterized above or below tight bounds. *J. Comp. Sys. Sci.*, 77(2):422 – 429, 2011.

15 R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Communications*, pages 85–103, 1972.

16 S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for Max-Cut and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

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

18 M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences*, 75:137–153, 2009.

19 M. Mnich, G. Philip, S. Saurabh, and O. Suchý. Beyond max-cut: λ-extendible properties parameterized above the Poljak-Turzík bound. *CoRR*, abs/1207.5696, 2012.

20 S. Poljak and D. Turzík. A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. *Discrete Mathematics*, 58(1):99–104, 1986.

21 V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446 – 458, 2006.

22 V. Raman and S. Saurabh. Improved fixed parameter tractable algorithms for two "edge" problems: MaxCut and MaxDag. *Information Processing Letters*, 104(2):65–72, 2007.

Subexponential Parameterized Odd Cycle Transversal on Planar Graphs

Daniel Lokshtanov[1], Saket Saurabh[2], and Magnus Wahlström[3]

1 **Department of Computer Science and Engineering, University of California, San Diego, USA.** daniello@ii.uib.no
2 **The Institute of Mathematical Sciences, Chennai, India.** saket@imsc.res.in
3 **Max-Planck-Institut für Informatik, Saarbrücken, Germany.** wahl@mpi-inf.mpg.de

Abstract

In the Odd Cycle Transversal (OCT) problem we are given a graph G on n vertices and an integer k, the objective is to determine whether there exists a vertex set O in G of size at most k such that $G \setminus O$ is bipartite. Reed, Smith and Vetta [Oper. Res. Lett., 2004] gave an algorithm for OCT with running time $3^k n^{O(1)}$. Assuming the exponential time hypothesis of Impagliazzo, Paturi and Zane, the running time can not be improved to $2^{o(k)}n^{O(1)}$. We show that OCT admits a randomized algorithm running in $O(n^{O(1)} + 2^{O(\sqrt{k}\log k)}n)$ time when the input graph is planar. As a byproduct we also obtain a linear time algorithm for OCT on planar graphs with running time $O(2^{O(k \log k)}n)$ time. This improves over an algorithm of Fiorini et al. [Disc. Appl. Math., 2008].

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph Theory, Parameterized Algorithms, Odd Cycle Transversal

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.424

1 Introduction

We consider the Odd Cycle Transversal (OCT) problem where we are given as input a graph G with n vertices and m edges, together with an integer k. The objective is to determine whether there exists a vertex set O of size at most k such that $G \setminus O$ is bipartite. This classical optimization problem was proven NP-complete already in 1978 by Yannakakis [28] and has been studied extensively both within approximation algorithms [1, 16] and parameterized algorithms [12, 17, 20, 22, 24, 26].

It was a long-standing open problem whether OCT is *fixed-parameter tractable (FPT)*, that is solvable in time $f(k)n^{O(1)}$ for some function f depending only on k. In 2004 Reed, Smith and Vetta [26] resolved the question positively, and gave an $O(4^k kmn)$ time algorithm for the problem. It was later observed by Hüffner [17] that the running time of the algorithm of Reed et al. is actually $O(3^k kmn)$.

Improving over the algorithm of Reed et al. [26], both in terms of the dependence on k, and in terms of the dependence on input size remain interesting research directions. For the dependence on input size, Reed et al. [26] point out that using techniques from the Graph Minors project of Robertson and Seymour one could improve the nm factor in the running time of their algorithm to n^2, at the cost of worsening the dependence on k. They pose the existence of a linear time algorithm for OCT for every fixed value of k as an open problem. Fiorini et al [12] showed that if the input graph is required to be planar, then OCT has a $2^{O(k^6)}n$ time algorithm. Later Kawarabayashi and Reed [20] gave an "almost" linear time

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 424–434

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

algorithm for OCT, that is an algorithm with running time $f(k)n\alpha(n)$ where $\alpha(n)$ is the inverse ackermann function and f is some computable function of k.

When it comes to the dependence of the running time on k, the $O(3^k nm)$ algorithm of Reed et al. [26] remained the best known until a recent manuscript of Lokshtanov et al. [24] (see also Narayanaswamy et al. [25]) giving an algorithm with running time $O(2.32^k n^{O(1)})$ using linear programming techniques. It is tempting to ask how far down one may push the dependence of the running time on k. Should we settle for c^k for a reasonably small constant c, or does there exist a *subexponential* parameterized algorithm for OCT, that is an algorithm with running time $2^{o(k)}n^{O(1)}$? It turns out that assuming the *Exponential Time Hypothesis* of Impagliazzo, Paturi and Zane [18] there can not be a subexponential parameterized algorithm for OCT. In this paper we show that restricting the input to planar graphs circumvents this obstacle – in particular we give an $O(n^{O(1)} + 2^{O(\sqrt{k}\log k)}n)$ time algorithm for OCT on planar graphs (we will refer to OCT on planar graphs as Pl-OCT). As a corollary of our main result we also obtain a simple $O(k^{O(k)}n)$ time algorithm for Pl-OCT, improving over the dependence on k in the algorithm of Fiorini et al. [12] while keeping the linear dependence on n.

Methods. There are many NP-complete graph problems that remain NP-complete even when restricted to planar graphs [15] but admit much better approximation algorithms and faster parameterized algorithms on planar graphs than on general graphs. The *bidimensionality theory* of Demaine et al. [7, 10] aims to explain this phenomenon. Specifically, using bidimensionality one can give fast parameterized algorithms [6], approximation schemes [8, 13] and efficient polynomial time pre-processing algorithms [14], called kernelization algorithms, for a host of problems on planar graphs, and more generally on classes excluding a forbidden minor. The main driving force behind bidimensionality is that for many parameterized problems on planar graphs one can bound the *treewidth* of the input graph as a *sublinear* function of the parameter k. For some problems, including OCT, this approach seems not to be amenable as there is no apparent connection between the parameter k and the treewidth of the input graph. Nevertheless, a variant of this idea is still the engine of the subexponential time parameterized algorithms of Dorn et al. [9] and Tazari [27], the linear time algorithm of Fiorini et al. [12] and also of our algorithm.

Fiorini et al. show that after a linear time pre-processing step, the treewidth of the input graph is bounded by $O(k^2)$. Well-known algorithms for finding tree decompositions [3] and an algorithm for OCT on graphs of bounded treewidth then do the job. To obtain our $O(k^{O(k)}n)$ time algorithm for Pl-OCT, we give a linear time *branching* step inspired by Baker's layering approach [2] that produces $O(k)$ instances, each of treewidth $O(k)$, such that the input instance is a "yes" instance if and only if at least one of the output instances is a "yes" instance. We then show that one can make a trade-off between the number of output instances of the branching process and the treewidth of the output graphs. In particular we show that we can output $k^{O(\sqrt{k})}$ instances each of treewidth $O(\sqrt{k})$, such that the input instance is a "yes" instance if and only if at least one of the output instances is a "yes" instance. The parameters involved in this trade-off are rather delicate, and to make the trade-off go through we need to first pre-process the graph using the recent sophisticated methods of Kratsch and Wahlström [22, 23]. This pre-processing step is the only part of our algorithm which takes superlinear time, and so we obtain an algorithm with running time $O(n^{O(1)} + 2^{O(\sqrt{k}\log k)}n)$. It remains an interesting open problem whether there is a subexponential parameterized algorithm for Pl-OCT with linear dependence on n.

2 Preliminaries

Throughout this paper we use n to denote the size of the vertex set of the input graph G. For a graph G we denote its vertex set by $V(G)$ and the edge set by $E(G)$. An edge between vertices u and v is denoted by uv, and is identical to the edge vu. We use $G[V']$ to denote the subgraph of G induced by V', i.e., the graph on vertex set V' and edge set $\{uv \in E(G) \mid u, v \in V'\}$. We use $G \setminus Z$ as an abbreviation for $G[V(G) \setminus Z]$. The open neighborhood of a vertex v in graph G contains the vertices adjacent to v, and is written as $N_G(v)$. The open neighborhood of a set $S \subseteq V(G)$ is defined as $\bigcup_{v \in S} N_G(v) \setminus S$. We omit the subscript G when it is clear from the context. A graph G is *bipartite* if there exists a partition of $V(G)$ into two sets A and B such that every edge of G has one endpoint in A and one in B. The sets A and B are called bipartitions of G. A set W of $V(G)$ is called an *odd cycle transversal* of G if $G \setminus W$ is bipartite. A *plane* embedding of a graph G is an embedding of G in the plane with no edge crossings. A graph G that has a plane embedding is called *planar*. A *plane* graph is a graph G together with a plane embedding of it. For a plane graph G, $F(G)$ is the set of faces of G.

2.1 Tree-width

Let G be a graph. A *tree decomposition* of a graph G is a pair $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$ (here T is a tree) such that

1. $\bigcup_{t \in V(T)} X_t = V(G)$,
2. for every edge $\{x, y\} \in E(G)$ there is a $t \in V(T)$ such that $\{x, y\} \subseteq X_t$, and
3. for every vertex $v \in V(G)$ the subgraph of T induced by the set $\{t \mid v \in X_t\}$ is connected.

The *width* of a tree decomposition is $\left(\max_{t \in V(T)} |X_t|\right) - 1$ and the *treewidth* of G is the minimum width over all tree decompositions of G. We use $\mathbf{tw}(G)$ to denote the treewidth of the input graph G.

A tree decomposition $(T, \mathcal{X})$ is called a *nice tree decomposition* if T is a tree rooted at some node r where $X_r = \emptyset$, each node of T has at most two children, and each node is of one of the following kinds:

1. *Introduce node*: a node t that has only one child t' where $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$.
2. *Forget node*: a node t that has only one child t' where $X_t \subset X_{t'}$ and $|X_t| = |X_{t'}| - 1$.
3. *Join node*: a node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.
4. *Leaf node*: a node t that is a leaf of t, is different than the root, and $X_t = \emptyset$.

Notice that, according to the above definition, the root r of T is either a forget node or a join node. It is well-known that any tree decomposition of G can be transformed into a nice tree decomposition in time $O(|V(G)| + |E(G)|)$ maintaining the same width [21]. We use G_t to denote the graph induced on the vertices $\bigcup_{t'} X'_t$, where t' ranges over all descendants of t, including t. We use H_t to denote $G_t[V(G_t) \setminus X_t]$.

3 Subexponential Time FPT Algorithm for Pl-OCT

In this section we outline our algorithms for Pl-OCT – (a) an algorithm running in time $O(k^{O(k)}n)$ and (b) an algorithm running in time $O(n^{O(1)} + 2^{O(\sqrt{k}\log k)}n)$. To do so we reduce the problem to a "Steiner tree-like" problem on graphs of small treewidth and then use an algorithm for this Steiner tree-like problem on graphs of bounded treewidth to obtain our results.

3.1 Reducting Pl-OCT to a "Steiner tree-like" problem

It is well-known that a plane graph is bipartite if and only if every face is even. Here we say that a face is even if the cyclic walk enclosing the face has even length. This fact allows us to interpret the OCT problem on a plane graph G as the "Steiner tree-like" L-Join problem on the face-vertex incidence graph of G. The *face-vertex incidence graph* of a plane graph G is the graph G^+ with vertex set $V(H) = V(G) \cup F(G)$ and an edge between a face $f \in F(G)$ and vertex $v \in V(G)$ if v is incident to f in the embedding of G. Clearly G^+ is planar, and also it is bipartite with bipartitions $V(G)$ and $F(G)$. For subsets $L \subseteq F(G)$ and $O \subseteq V(G)$ we will say that O is an *L-join* in G^+ if every connected component of $G^+[F(G) \cup O]$ contains an even number of vertices from L. The following observation plays a crucial role in our algorithm.

▶ **Proposition 1** ([12]). A subset O of $V(G)$ is an odd cycle transversal of G if and only if every connected component of $G^+[F(G) \cup O]$ has an even number of vertices of L. Here, L is the set of odd faces of G.

Observe that the notion of an L-join can be defined for any bipartite graph H with bipartitions A and B. Specifically for subsets $L \subseteq A$ and $O \subseteq B$ we say that O is an L-join in H if every connected component of $H[A \cup O]$ contains an even number of vertices from L. In the L-Join problem we are given a bipartite graph H with bipartitions A and B, together with a subset $L \subseteq A$ and an integer k. The task is to determine whether there is an L-join $W \subseteq B$ in H of size at most k. The Pl-L-Join problem is just L-Join, but with the input graph H required to be planar. Proposition 1 directly implies the following lemma.

▶ **Lemma 2.** *If there is an algorithm for* Pl-L-Join *with running time $O(f(k)n^c)$ for a function f and constant $c \geq 1$ then there is an algorithm for* Pl-OCT *with running time $O(f(k)n^c)$.*

In Section 3.2 we will give an algorithm for Pl-L-Join with running time $O(2^{O(k \log k)}n)$, yielding an algorithm for Pl-OCT with the same running time. To get a subexponential time algorithm for Pl-OCT we will reduce to a *promise* variant of Pl-L-Join where we additionally are given a set S of size $k^{O(1)}$ with the promise that an optimal solution can be found inside S. We now formally define the promise variant of Pl-L-Join that we will reduce to.

Promise Planar-L-Join (PrPl-L-Join)	
Input:	A bipartite planar graph H with bipartitions A and B, a set of terminals $L \subseteq A$, a set of annotated vertices $S \subseteq B$ and an integer k
Parameter:	$\|S\|$, k
Question:	Is there an L-join $O \subseteq B$ of size at most k?
Promise:	If an L-join $O \subseteq B$ of size at most k exists then there is an L-join $O' \subseteq S$ of size at most $\|O\|$.

In order to be able to reduce Pl-OCT to PrPl-L-Join we show the following lemma.

▶ **Lemma 3** (Small Relevant Set Lemma). *Let (G, k) be a yes instance to* Pl-OCT. *Then in polynomial time we can find a set S such that*

- *$|S| = k^{O(1)}$; and*
- *with probability $(1 - \frac{1}{2^n})$, G has an oct of size k if and only if there is an oct contained in S of size k.*

Here $n = |V(G)|$.

Proof. This follows from [22, 23], but for completeness we sketch the proof here. First, we find in polynomial time an approximate solution of size at most $\frac{9}{4}k$ by applying the $\frac{9}{4}$-approximation algorithm for Pl-OCT by Goemans and Williamson [16]. Let X be such an approximate solution. Next, we create an auxiliary graph G' from G and X as in the algorithm of Reed, Smith, and Vetta [26]; the vertex set of G' is $(V \setminus X) \cup X'$, where X' is a set of $2|X|$ terminals corresponding to X. It is a consequence of [26], made explicit in [22, Lemma 4.1], that a minimum oct can be found by taking the union of a subset of X and a minimum S-T vertex cut in $G' \setminus R$ for $S, T, R \subseteq X'$ (it may be assumed that all minimum cuts are disjoint from X', by modifying R). By [23, Corollary 1], there exists a set $Z \subseteq V(G')$ with $|Z| = O(|X|^3)$ which includes such a min-cut for all choices of S, T, R, and we can find it in polynomial time, with success probability as stated, using the tools of representative sets from matroid theory; see [23]. ◀

Proposition 1 together with Lemma 3 directly imply the following lemma.

▶ **Lemma 4.** *If there is an algorithm for* PrPl-L-Join *with running time $O(f(k)n^c)$ for a function f and constant $c \geq 1$ then there is a randomized algorithm for* Pl-OCT *with running time $O(n^{O(1)} + f(k)n^c)$ and success probability at least $(1 - \frac{1}{2^n})$.*

At this point we make a remark about results in [22, 23]. In [22, 23], Kratsch and Wahlström obtain a polynomial kernel for OCT. That is, given an input (G, k) they output an equivalent instance (G', k') such that G has an odd cycle transversal of size k if and only if G' has and $k' \leq k$. It is very tempting to use this result directly at the place of Lemma 3. However, for our subexponential algorithm for Pl-OCT we not only need that $k' \leq k$, G' has small size but also that G' is a planar graph. However, it is not clear that the algorithms described in [22, 23] could be easily modified to get both $k' \leq k$ and G' is planar. Thus we resort to Lemma 3 which is sufficient for our purpose.

3.2 Algorithms for Pl-L-Join, PrPl-L-Join and Pl-OCT

In this section we will give fast parameterized algorithms for Pl-L-Join and PrPl-L-Join. The algorithms are based on the following decomposition lemma.

▶ **Lemma 5.** *There is an algorithm that given a planar bipartite graph H with bipartitions A and B and an integer t, runs in time $O(n)$ and computes a partition of B into $B = B_1 \cup B_2 \ldots \cup B_t$ such that $\mathbf{tw}(G \setminus B_i) = O(t)$ for every $i \leq t$. Furthermore, for every $i \leq t$ a tree-decomposition of $G \setminus B_i$ of width $O(t)$ can be computed in time $O(tn)$.*

Proof. Select a vertex $r \in A$ and do a breadth first search in H starting from r. We call $\{r\}$ the first BFS layer, $N(r)$ the second BFS layer, $N(N(r)) \setminus \{r\}$ the third BFS layer etc. Let $L_1, L_2, \ldots, L_\ell$ be the BFS layers of H. Since H is bipartite we have that for every odd i, $L_i \subseteq A$ while for every even i we have $L_i \subseteq B$. For every i from 1 to t set $B_i = \bigcup_{j \geq 0} L_{2i+2tj}$. It is easy to see that $B_1, \ldots, B_t$ indeed form a partition of B. Furthermore, for every i, every connected component C of $H \setminus B_i$ is a subset of at most $2t$ consecutive BFS layers of H. Contracting all of the BFS layers preceeding C in H into a single vertex shows that C is an induced subgraph of a planar graph of diameter $O(t)$. Thus it follows from [4, 11] that a tree decomposition of C of width $O(t)$ can be computed in time $O(t|C|)$. Hence for every $i \leq t$ a tree-decomposition of $G \setminus B_i$ of width $O(t)$ can be computed in time $O(tn)$. ◀

In Section 4 we will prove the following lemma.

▶ **Lemma 6.** *There is an algorithm that given an bipartite graph H with bipartitions A and B, together with a set $L \subseteq A$, an integer k and a tree-decomposition of H of width w, determines whether there is an L-join $W \subseteq B$ of size at most k in time $O(w^{O(w)}n)$.*

Lemmata 5 and 6 yield the $O(2^{O(k \log k)}n)$ time algorithm for Pl-L-Join.

▶ **Lemma 7.** *There is a $O(2^{O(k \log k)}n)$ time algorithm for* Pl-L-Join.

Proof. Given as input a planar bipartite graph H with bipartitions A and B, a set $L \subseteq A$ and an integer k the algorithm applies Lemma 5 with $t = k + 1$. Now, if H has an L-join W of size at most k then there is an $i \leq t$ such that $W \cap B_i = \emptyset$, and so W is an L-join in $H \setminus B_i$. Furthermore, for any j an L-join in $H \setminus B_j$ is also an L-join in H. We loop over every i and return the smallest L-join of $H \setminus B_i$. By Lemma 5, for each i we can compute a tree-decomposition of $H \setminus B_i$ of width $O(t)$ in $O(tn)$ time. By Lemma 6 we can find a smallest L-join of $H \setminus B_i$ in time $O(2^{O(k \log k)}n)$. ◀

The algorithm for PrPl-L-Join goes along the same lines as the algorithm in Lemma 7, but is slightly more involved.

▶ **Lemma 8.** *There is an $O(|S|^{\sqrt{k}} \cdot 2^{O(\sqrt{k} \log k)} \cdot n)$ time algorithm for* PrPl-L-Join.

Proof. Given as input a planar bipartite graph H with bipartitions A and B, a set $L \subseteq A$ of terminals and a set $S \subseteq B$ of annotated vertices together with an integer k the algorithm applies Lemma 5 with $t = \sqrt{k}$. For every $i \leq t$ define $W_i = W \cap B_i$. Now, if H has an L-join W of size at most k then without loss of generality $W \subseteq S$. Furthermore there is an $i \leq t$ such that $|W_i| \leq \sqrt{k}$. Observe that W is also an L-join in $H \setminus (B_i \setminus W_i)$. Furthermore, for any subset B' of B, an L-join in $H \setminus B'$ is also an L-join in H. The algorithm loops over every i, and every choice of $W_i^* \subseteq B_i \cap S$ with $|W_i^*| \leq \sqrt{k}$. There are $\sqrt{k}$ choices for i and at most $|S|^{\sqrt{k}}$ choices for W_i^*. For each choice of i and W_i^* the algorithm finds the smallest L-join of $H \setminus (B_i \setminus W_i^*)$. Correctness follows from the fact that we will loop over the choice $W_i^* = W_i$.

In order to find the smallest L-join of $H \setminus (B_i \setminus W_i^*)$ we will apply Lemma 6, but in order to do that we need a tree decomposition of $H \setminus (B_i \setminus W_i^*)$ of small width. However, by Lemma 5 we can find a tree decomposition of $H \setminus B_i$ of width $O(\sqrt{k})$ in linear time for every i. Adding W_i^* to every bag of this tree decomposition yields a tree decomposition of $H \setminus (B_i \setminus W_i^*)$ of width $O(\sqrt{k}) + |W_i^*| = O(\sqrt{k})$. Thus, by Lemma 6 we can find the smallest L-join of $H \setminus (B_i \setminus W_i^*)$ in time $O(2^{O(\sqrt{k} \log k)} \cdot n)$ for every choice of i and W_i^*. Since there are $|S|^{\sqrt{k}}$ choices for W_i and $\sqrt{k}$ choices for i this concludes the proof. ◀

We are now ready to prove our main theorems. In particular, Lemmata 2 and 7 imply our linear time parameterized algorithm for Pl-OCT.

▶ **Theorem 9.** *There is a $O(2^{O(k \log k)}n)$ time algorithm for* Pl-OCT.

Similarly, Lemmata 4 and 8 imply our subexponential parameterized algorithm for Pl-OCT.

▶ **Theorem 10.** *There is an $O(n^{O(1)} + 2^{O(\sqrt{k} \log k)}n)$ time randomized algorithm for* Pl-OCT.

4 An algorithm for Minimum L-Join on graphs of bounded treewidth

In this section we give a dynamic programming algorithm on graphs of bounded treewidth for the following problem.

Minimum L-Join	
Input:	A bipartite graph G with bipartitions C and D and a set $L \subseteq C$.
Parameter:	$\mathbf{tw}(G)$
Question:	Find a minimum sized set $W \subseteq D$ (if it exists) such that every connected component of $G[C \cup W]$ has an even number of vertices of L.

Observe that finding W is equivalent to finding a forest F of G such that $L \subseteq V(F)$ and each tree of F contains an even number of vertices of L.

4.1 Description of the Algorithm

The idea of our algorithm is to do dynamic programming starting from leaf to root. We set

$$|X_t| = w \qquad L_t = L \cap V(G_t) \qquad C_t = C \cap V(G_t)$$

For a node t and any solution, say F, intersection with G_t and X_t (a partial solution) could be described as follows:

- A tree F_i of F is contained inside $V(G_t) \setminus X_t$ and in this case we have that $|V(F_i) \cap L|$ is even.
- A tree F_i of F does not contain any vertex of $V(G_t)$.
- A tree F_i of F contains vertices from both $V(G_t)$ and $V(G) \setminus V(G_t)$. In this case we have that F_i contains vertices from X_t and either contains an even or an odd number of vertices from L.

We would like to keep representatives for all partial solutions for the graph G_t. Towards this we first introduce the following definition.

▶ **Definition 11.** A set P is a *partition* of X if, and only if, it does not contain the empty set unless $X = \emptyset$ and: (a) the union of the elements of P is equal to X; and (b) the intersection of any two elements of P is empty. (We say the elements of P are pairwise disjoint.) We call an element of P as *piece*. A partition is called *signed partition* if for every piece $A \in P$, we assign either 0 or 1. The sign of a piece A is denoted by $sign(A)$. That is, $sign$ is a function from P to $\{0, 1\}$. A signed partition is denoted by $(P, sign)$, that is, a pair consisting of the partition P and a function $sign : P \to \{0, 1\}$.

For each node $i \in V(T)$ we compute a table A_i, the rows of which are 3-tuples $[S, (P, sign), val]$. Table A_i contains one row for each combination of the first two components which denote the following:

- S is a subset of X_i.
- $(P, sign)$, where P is a partition of S into at most $|S|$ labelled pieces.

We use $P(v)$ to denote the piece of the partition P that contains the vertex v. We let $|P|$ denote the number of pieces in the partition P. The set S denotes the intersection of our solution with the vertices in the bag X_i.

The last component val, also denoted as $A_i[S, (P, sign)]$, is the size of a smallest forest $F_i(S, (P, sign))$ of G_i which satisfies the following properties:

- $C_i \subseteq V(F_i(S, (P, sign)))$ – all the vertices of C lying in G_i are contained in the forest;
- $(X_i \setminus S) \cap V(F_i(S, (P, sign))) = \emptyset$ – only vertices in S from X_i are contained in the forest;
- for every non-empty part A of P there exists a tree, say F_A in $F_i(S, (P, sign))$, such that $A \subseteq V(F_A)$ and $|L_i \cap V(F_A)| \bmod 2 = sign(A)$ and for every $A \neq B$, $F_A \neq F_B$ (that is, trees associated with distinct parts are distinct); and

- if there exists a tree F'' in $F_i(S,(P,sign))$ such that $V(F'') \cap X_i = \emptyset$ then $|L_i \cap V(F'')| \bmod 2 = 0$.

If there is no such forest $F_i(S,(P,sign))$, then the last component of the row is set to ∞. Given a node i of the tree T and a pair $(S,(P,sign))$ of X_i, a forest F in G_i satisfying the above properties is called *consistent* with $(S,(P,sign))$.

We compute the tables A_i starting from the leaf nodes of the tree decomposition and going up to the root.

Leaf Nodes. Let i be a leaf node of the tree decomposition. We compute the table A_i as follows. We set $A_i[\emptyset,(\emptyset,0)] = 0$ and $A_i[\emptyset,(\emptyset,1)] = 0$.

Introduce Nodes. Let i be an introduce node and j its unique child. Let $x \in X_i \setminus X_j$ be the introduced vertex. For each pair $(S,(P,sign))$, we compute the entry $A_i[S,(P,sign)]$ as follows.

Case 1. $x \in S$. Check whether $N(x) \cap S \subseteq P(x)$; if not, set $A_i[S,(P,sign)] = \infty$.

Subcase 1: $P(x) = \{x\}$. If ($x \in L_i$ and $sign(P(x)) = 0$) or ($x \notin L_i$ and $sign(P(x)) = 1$) then set $A_i[S,(P,sign)] = \infty$.
Else, we set $A_i[S,(P,sign)] = A_j[S \setminus \{x\},(P \setminus P(x),sign')] + 1$. Here $sign'$ is the restriction of $sign$ to $P \setminus P(x)$.

Subcase 2: $|P(x)| \geq 2$ and $N(x) \cap P(x) = \emptyset$. Set $A_i[S,(P,sign)] = \infty$, as no extension of $P(x)$ in G_i is connected.

Subcase 3: $|P(x)| \geq 2$ and $N(x) \cap P(x) \neq \emptyset$. Let $\mathcal{A}$ be the set of all rows $[S',(P',sign')]$ of the table A_j that satisfy the following conditions:

- $S' = S \setminus \{x\}$.
- $P' = (P \setminus P(x)) \cup Q$, where Q is a partition of $P(x) \setminus \{x\}$ such that each piece of Q contains an element of $N(x) \cap P(x)$.
- $sign'$ is such that it agrees with $sign$ on $P \setminus P(x)$ and if $x \in L_i$ then

$$\left(1 + \sum_{Q_\ell \in Q} sign'(Q_\ell)\right) \bmod 2 = sign(P(x),$$

else

$$\left(\sum_{Q_\ell \in Q} sign'(Q_\ell)\right) \bmod 2 = sign(P(x)).$$

Set $A_i[S,(P,sign)] = \min_{[S',(P',sign')] \in \mathcal{A}} \left\{A_j[S',(P',sign')]\right\} + 1$.

Case 2. $x \notin S$. If $x \in C_i$ then set $A_i[S,(P,sign)] = \infty$. Else set $A_i[S,(P,sign)] = A_j[S,(P,sign)]$.

Forget Nodes. Let i be a forget node and j its unique child node. Let $x \in X_j \setminus X_i$ be the forgotten vertex. For each pair $(S,(P,sign))$ in the table A_i, let $\mathcal{A}$ be the set of all rows $[S',(P',sign')]$ of the table A_j that satisfy the following conditions:

- $S' = S \cup \{x\}$, and
- $P'(x) = P(y) \cup \{x\}$ for some $y \in S$ and all other parts remain the same. Essentially, P' has been obtained by adding x to some part of P.
- $sign'$ is same as $sign$ on all other parts of P' but $P'(x)$ and $sign(P'(x)) = sign(P(y))$.

Set

$$A_i[S,(P,sign)] = \min_{[S',(P',sign')]\in\mathcal{A}} \Big\{ A_j[S',(P',sign')] \Big\}.$$

Join Nodes. Let i be a join node and j and l its children. For each triple $(S,(P,sign))$ we compute $A_i[S,(P,sign)]$ as follows.
Let $\mathcal{A}$ denote the set of all pairs $\langle (S,(P_1,sign_1)),(S,(P_2,sign_2))\rangle$, where $(S,(P_1,sign_1)) \in A_j$ and $(S,(P_2,sign_2)) \in A_l$ with the following property:

Starting with the partitions $Q_p = P_1$ and the sign function $sign_p = sign_1$ and repeatedly applying the following operation, we reach the stable partition that is identical to $(P,sign)$. The operation that we apply is:

> If there exist vertices $u,v \in S$ such that they are in different pieces of Q_p but are in the same piece of P_2, delete $Q_p(u)$ and $Q_p(v)$ from Q_p and add $Q_p(u) \cup Q_p(v)$. Furthermore make $sign_p(Q_p(u) \cup Q_p(v)) := (sign_p(P(u)) + sign_p(P(v))) \bmod 2$.

Set

$$A_i[S,(P,sign)] = \min_{\langle (S,(P_1,sign_1)),(S,(P_2,sign_2))\rangle\in\mathcal{A}} \Big\{ A_j[S,(P_1,sign_1)] + A_l[S,(P_2,sign_2)] - |S| \Big\}.$$

The stated conditions ensure that $u,v \in S$ are in the same piece of P if and only if for each $\langle (S,(P_1,sign_1)),(S,(P_2,sign_2))\rangle \in \mathcal{A}$, they are in the same piece of P_1 or of P_2 (or both). Given this, it is easy to verify that the above computation correctly determines $A_i\,[S,(P,sign)]$.

Root Node. We obtain the size of a smallest L-join of G from any row of the table A_r for the root node r. That is, if the size of the forest we have stored is η, then the size of the smallest L-join of G is $\eta - |C|$.

Extracting the solution at the root node. We can compute the optimum solution, that is the set W, by standard backtracking or by storing a set of vertices for each row and each bag.

4.2 Correctness and the Time analysis of the algorithm

We are now ready to discuss the algorithm's running time and prove that it correctly computes an optimal solution.

Proof. (of Lemma 6) We first upper-bound the running time of the algorithm we described earlier. The running time mainly depends on the size of the tables and the combination of tables during the bottom-up traversal of the decomposition tree. Let ζ be the size of the number of signed partitions of size at most $w+1$. The number ζ is upper bounded by $(w+1)^{w+1} \times 2^{w+1}$. Thus the size of the table at any node is upper bounded by $2^{w+1} \times \zeta = 4^{w+1}(w+1)^{w+1} = w^{O(w)}$. Furthermore time taken to compute the value for any row is upper bounded by $w^{O(w)}$. Thus the total time taken by the algorithm is upper bounded by $w^{O(w)} \cdot n = 2^{O(w \log w)} \cdot n$.

The algorithm's correctness can be shown by a standard inductive proof on the decomposition tree. This completes the proof. For an example see [5] for similar proof for the Steiner tree problem parameterized by treewidth of the input graph. ◀

5 Open Problems and Conclusions

In this paper we gave the first subexponential time algorithm for Pl-OCT combining the recent matroid based kernelization for OCT and a reformulation of Pl-OCT in terms of T-joins. On the way we also obtained an algorithm for Pl-OCT running in time $O(k^{O(k)}n)$, improving over the previous linear time FPT algorithm for Pl-OCT by Fiorini et al. [12]. Let us remark that Fiorini et al. [12] do not compute the dependence of the running time on k of their algorithm, and the running time of their algorithm depends on how one implements a particular step, where one needs to compute a tree-decomposition of width $O(k^2)$ of a particular planar graph. Naively using Bodlaender's algorithm [3] gives an $O(2^{O(k^6)}n)$ time algorithm. By using more clever tricks, such as using Kammer and Tholey's [19] recent linear time constant factor approximation algorithm for treewidth of planar graphs, one may get an $O(2^{O(k^2)}n)$ time algorithm. This is still quite a bit slower than our $O(k^{O(k)}n)$ running time.

We conclude with two interesting problems that remain open. First, is there a subexponential parameterized algorithm for Pl-OCT with linear dependence on n? Second, is there an algorithm for Pl-OCT running in time $2^{O(\sqrt{k})}n^{O(1)}$?.

References

1 A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In *STOC*, pages 573–581, 2005.

2 B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

3 H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

4 H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.

5 M. Chimani, P. Mutzel, and B. Zey. Improved Steiner tree algorithms for bounded treewidth. In *IWOCA*, volume 7056, pages 374–386, 2011.

6 E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *J. ACM*, 52(6), 2005.

7 E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

8 E. D. Demaine and M. T. Hajiaghayi. Bidimensionality: new connections between FPT algorithms and PTASs. In *SODA*, pages 590–601, 2005.

9 F. Dorn, F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. In *STACS*, pages 251–262, 2010.

10 F. Dorn, F. V. Fomin, and D. M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

11 D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

12 S. Fiorini, N. Hardy, B. A. Reed, and A. Vetta. Planar graph bipartization in linear time. *Discrete Applied Mathematics*, 156(7):1175–1180, 2008.

13 F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Bidimensionality and EPTAS. In *SODA*, pages 748–759, 2011.

14 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In *SODA*, pages 503–510, 2010.

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

16 M. X. Goemans and D. P. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica*, 18(1):37–59, 1998.

17 F. Hüffner. Algorithm engineering for optimal graph bipartization. *J. Graph Algorithms Appl.*, 13(2):77–98, 2009.

18 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

19 F. Kammer and T. Tholey. Approximate tree decompositions of planar graphs in linear time. In *SODA*, pages 683–698, 2012.

20 K. Kawarabayashi and B. A. Reed. An (almost) linear time algorithm for odd cyles transversal. In *SODA*, pages 365–378, 2010.

21 T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

22 S. Kratsch and M. Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In *SODA*, pages 94–103, 2012.

23 S. Kratsch and M. Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS*, 2012. To appear. Preprint available at `http://arxiv.org/abs/1111.2195`.

24 D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. Faster parameterized algorithms using linear programming. *CoRR*, abs/1203.0833, 2012.

25 N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. LP can be a cure for parameterized problems. In *STACS*, pages 338–349, 2012.

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

27 S. Tazari. Faster approximation schemes and parameterized algorithms on h-minor-free and odd-minor-free graphs. In *MFCS*, pages 641–652, 2010.

28 M. Yannakakis. Node- and edge-deletion NP-complete problems. In *STOC*, pages 253–264, 1978.

Deciding Probabilistic Automata Weak Bisimulation in Polynomial Time

Holger Hermanns and Andrea Turrini

Saarland University – Computer Science, Saarbrücken, Germany

Abstract

Deciding in an efficient way weak probabilistic bisimulation in the context of probabilistic automata is an open problem for about a decade. In this work we close this problem by proposing a procedure that checks in polynomial time the existence of a weak combined transition satisfying the step condition of the bisimulation. This enables us to arrive at a polynomial time algorithm for deciding weak probabilistic bisimulation. We also present several extensions to interesting related problems setting the ground for the development of more effective and compositional analysis algorithms for probabilistic systems.

1998 ACM Subject Classification G.3: Probability and Statistics; F.2: Analysis Of Algorithms and Problem Complexity.

Keywords and phrases Probabilistic Automata, Weak probabilsitic bisimulation, Linear Programming problem, Polynomial decision algorithm.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.435

1 Introduction

Probabilistic automata (*PA*) constitute a mathematical framework for the specification of probabilistic concurrent systems [4,22]. Probabilistic automata extend classical concurrency models in a simple yet conservative fashion. In probabilistic automata, there is no global notion of time, and probabilistic experiments can be performed inside a transition. This embodies a clear separation between probability and nondeterminism, and is represented by transitions of the form $s \xrightarrow{a} \mu$, where s is a state, a is an action label, and μ is a probability distribution on states. Labeled transition systems are instances of this model family, obtained by restricting to Dirac distributions (assigning full probability to single states). Thus, foundational concepts and results of standard concurrency theory are retained in full and extend smoothly to the model of probabilistic automata. The *PA* model is akin to Markov decision processes (*MDP*) [7], and its foundational beauty can be paired with powerful model checking techniques, as implemented for instance in the PRISM tool [16]. Variations of this model are Labeled Concurrent Markov Chains (*LCMC*) and alternating Models [11, 21, 27]. We refer the interested reader to [23] for a survey on *PA* and other models.

If facing a concrete probabilistic system, we can conceive several different *PA* models to reflect its behavior. For instance, we can use different state names, encode diverse information in the states, represent internal computations with different action labels, and so on. *Bisimulation relations* constitute a powerful tool allowing us to check whether two models describe essentially the same system. They are then called bisimilar. The bisimilarity of two systems can be viewed in terms of a game played between a challenger and a defender. In each step of the infinite bisimulation game, the challenger chooses one automaton, makes a step, and the defender matches it with a step of the other automaton. Depending on how we want to treat internal computations, this leads to *strong* and *weak* bisimulations: the

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 435–447

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

former requires that each single step of the challenger automaton is matched by an equally labeled single step of the defender automaton, the latter allows the matching up to internal computation steps. On the other hand, depending on how nondeterminism is resolved, probabilistic bisimulation can be varied by allowing the defender to match the challenger's step by a convex combination of enabled probabilistic transitions. This results in a spectrum of four bisimulations: strong [11, 22, 27], strong probabilistic [22], weak [21, 22], and weak probabilistic [22] bisimulation.

Besides comparing automata, bisimulation relations allow us to reduce the size of an automaton without changing its properties (i.e., with respect to logic formulae satisfied by it). This is particularly useful to alleviate the state explosion problem notoriously encountered in model checking.

Polynomial decision algorithms for strong (probabilistic) bisimulation [3] and weak bisimulation [21] are known. However, *PA* weak bisimulation lacks in transitivity and this severely limits its usefulness. On the other hand weak probabilistic bisimulation is indeed transitive, while the only known algorithm for such bisimulation is exponential [3] in the size of the probabilistic automaton.

In this context, it is worth to note that *LCMC* weak bisimulation [21] and *PA* weak probabilistic bisimulation [22] coincide [24] when *LCMC* is seen as a *PA* with restrictions on the structure of the automaton and that restricted versions of *PA* weak probabilistic bisimulations, such as normed [1] and delay [25] bisimulation, can be decided in polynomial time. Following [24], an *LCMC* is just a *PA* where each state with outgoing transitions enables either labeled transitions each one leading to a single state, or a single transition leading to a probability distribution over states and this constraint on the structure of the automaton is enough to reduce the complexity of the decision procedure at the expense of the loss of using combined transitions and nondeterminism to simplify the automaton.

Lately, the model of *PA* has been enhanced with memoryless continuous time, integrated into the model of Markov automata [6, 8, 9]. This extension is also rooted in interactive Markov chains (*IMC*) [13], another model with a well-understood compositional theory. *IMC*s are applied in a large spectrum of practical applications, ranging from networked hardware on chips [5] to water treatment facilities [12] and ultra-modern satellite designs [10]. The standard analysis trajectory for *IMC* revolves around compositional applications of weak bisimulation minimization, a strategy that has been proven very effective [2, 5, 14], and is based on a polynomial time weak bisimulation decision algorithm [13, 28]. Owed to the unavailability of effective algorithms for *PA* weak probabilistic bisimulations, this compositional minimization strategy has thus far not been applied in the *PA* (or *MDP*) setting. We aim at making this possible, and furthermore, we intend to repeat and extend the successful applications of *IMC* in the extended Markov automata setting. For this, a polynomial time decision procedures for weak probabilistic bisimulation on *PA* is the essential building block.

In this paper we show that *PA* weak probabilistic bisimulation can be decided in polynomial time, thus just as all other bisimulations on *PA*. To arrive there, we provide a decision procedure that follows the standard partition refinement approach [3, 17, 19] and that is based on a Linear Programming (LP) problem. The crucial step is that we manage to generate and decide an LP problem that proves or disproves the existence of a weak step in time polynomial in the size of an automaton which in turn encodes a weak transition linear in its size. This enables us to decide in polynomial time whether the defender has a matching weak transition step - opposed to the exponential time required thus far [3] for this. Apart from this result, which closes successfully the open problem of [3], we show how our LP

approach can be extended to hyper-transitions (weak transitions leaving a probability distribution instead of a single state) and to the novel concepts of allowed weak/hyper-transitions (weak/hyper-transitions involving only a restricted set of transitions) and of equivalence matching (given two states, check whether each one enables a weak transition matchable by the other). Hyper-transitions naturally occur in weak probabilistic bisimulation on Markov automata, and in the bisimulation formulation of probabilistic forward simulation [8, 22].

Organization of the paper. After the preliminaries in Section 2, we present in Section 3 the polynomial LP problem that models weak transitions together with several extensions that can be computed in polynomial time as well. Then, in Section 4, we recast the algorithm proposed in [3] that decides whether two probabilistic automata are weak probabilistic bisimilar and we show that the decision procedure is polynomial. We conclude the paper in Section 5 with some remarks. Due to space limitations, we refer the reader interested in detailed proofs to [15].

2 Mathematical Preliminaries

For a generic set X, denote by $\mathrm{Disc}(X)$ the set of discrete probability distributions over X, and by $\mathrm{SubDisc}(X)$ the set of discrete sub-probability distributions over X. Given $\rho \in \mathrm{SubDisc}(X)$, we denote by $\mathrm{Supp}(\rho)$ the set $\{\, x \in X \mid \rho(x) > 0 \,\}$, by $\rho(\bot)$ the value $1-\rho(X)$ where $\bot \notin X$, and by δ_x the *Dirac* distribution such that $\rho(x) = 1$ for $x \in X \cup \{\bot\}$. For a sub-probability distribution ρ, we also write $\rho = \{\, p_x x \mid x \in X, p_x = \rho(x) \,\}$. The lifting $\mathcal{L}(\mathcal{R})$ [18] of a relation $\mathcal{R} \subseteq X \times Y$ is defined as follows: for $\rho_X \in \mathrm{Disc}(X)$ and $\rho_Y \in \mathrm{Disc}(Y)$, $\rho_X \; \mathcal{L}(\mathcal{R}) \; \rho_Y$ holds if there exists a *weighting function* $w \colon X \times Y \to [0,1]$ such that (1) $w(x,y) > 0$ implies $x \; \mathcal{R} \; y$, (2) $\sum_{y \in Y} w(x,y) = \rho_X(x)$, and (3) $\sum_{x \in X} w(x,y) = \rho_Y(y)$. When $\mathcal{R}$ is an equivalence relation on a set X, $\rho_1 \; \mathcal{L}(\mathcal{R}) \; \rho_2$ holds if for each $\mathcal{C} \in X/\mathcal{R}$, $\rho_1(\mathcal{C}) = \rho_2(\mathcal{C})$.

A Probabilistic Automaton (*PA*) $\mathcal{A}$ is a tuple $(S, \bar{s}, \Sigma, D)$, where S is a set of *states*, $\bar{s} \in S$ is the *start state*, Σ is the set of *actions*, and $D \subseteq S \times \Sigma \times \mathrm{Disc}(S)$ is a *probabilistic transition relation*. The set Σ is parted in two sets H and E of internal (hidden) and external actions, respectively; we let s,t,u,v, and their variants with indices range over S, a, b range over actions, and τ range over hidden actions. In this work we consider only finite *PA*s, i.e., automata such that S and D are finite.

A transition $tr = (s, a, \mu) \in D$, also denoted by $s \xrightarrow{a} \mu$, is said to *leave* from state s, to be *labeled* by a, and to *lead* to μ, also denoted by μ_{tr}. We denote by $src(tr)$ the *source* state s, by $act(tr)$ the *action* a, and by $trg(tr)$ the *target* distribution μ. We also say that s enables action a, that action a is enabled from s, and that (s, a, μ) is enabled from s. Finally, we denote by $D(s)$ the set of transitions enabled from s, i.e., $D(s) = \{\, tr \in D \mid src(tr) = s \,\}$, and similarly by $D(a)$ the set of transitions with action a, i.e., $D(a) = \{\, tr \in D \mid act(tr) = a \,\}$.

An *execution fragment* of a *PA* $\mathcal{A}$ is a finite or infinite sequence of alternating states and actions $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$ starting from a state s_0, also denoted by $first(\alpha)$, and, if the sequence is finite, ending with a state, also denoted by $last(\alpha)$, such that for each $i > 0$ there exists a transition $(s_{i-1}, a_i, \mu_i) \in D$ such that $\mu_i(s_i) > 0$. The *length* of α, denoted by $|\alpha|$, is the number of occurrences of actions in α. If α is infinite, then $|\alpha| = \infty$. Denote by $frags(\mathcal{A})$ the set of execution fragments of $\mathcal{A}$ and by $frags^*(\mathcal{A})$ the set of finite execution fragments of $\mathcal{A}$. An execution fragment α is a *prefix* of an execution fragment α', denoted by $\alpha \leqslant \alpha'$, if the sequence α is a prefix of the sequence α'. The *trace* of α, denoted by $trace(\alpha)$, is the sub-sequence of external actions of α. For instance, for $a \in E$, $trace(s_0 a s_1) = trace(s_0 \tau s_1 \tau \ldots \tau s_{n-1} a s_n) = a$, also denoted by $trace(a)$, and

$trace(s_0) = trace(s_0 \tau s_1 \tau \dots \tau s_n) = \varepsilon$, the empty sequence, also denoted by $trace(\tau)$.

A *scheduler* for a PA $\mathcal{A}$ is a function $\sigma\colon frags^*(\mathcal{A}) \to \text{SubDisc}(D)$ such that for each finite execution fragment α, $\sigma(\alpha) \in \text{SubDisc}(D(last(\alpha)))$. A scheduler is *determinate* [3] if for each pair of execution fragments α, α', if $trace(\alpha) = trace(\alpha')$ and $last(\alpha) = last(\alpha')$, then $\sigma(\alpha) = \sigma(\alpha')$. Given a scheduler σ and a finite execution fragment α, the distribution $\sigma(\alpha)$ describes how transitions are chosen to move on from $last(\alpha)$. A scheduler σ and a state s induce a probability distribution $\mu_{\sigma,s}$ over execution fragments as follows. The basic measurable events are the cones of finite execution fragments, where the cone of a finite execution fragment α, denoted by C_α, is the set $\{\, \alpha' \in frags^*(\mathcal{A}) \mid \alpha \leqslant \alpha' \,\}$. The probability $\mu_{\sigma,s}$ of a cone C_α is defined recursively as follows:

$$\mu_{\sigma,s}(C_\alpha) = \begin{cases} 0 & \text{if } \alpha = t \text{ for a state } t \neq s, \\ 1 & \text{if } \alpha = s, \\ \mu_{\sigma,s}(C_{\alpha'}) \cdot \sum_{tr \in D(a)} \sigma(\alpha')(tr) \cdot \mu_{tr}(t) & \text{if } \alpha = \alpha' a t. \end{cases}$$

Standard measure theoretical arguments ensure that $\mu_{\sigma,s}$ extends uniquely to the σ-field generated by cones. We call the measure $\mu_{\sigma,s}$ a *probabilistic execution fragment* of $\mathcal{A}$ and we say that it is generated by σ from s. Given a finite execution fragment α, we define $\mu_{\sigma,s}(\alpha)$ as $\mu_{\sigma,s}(\alpha) = \mu_{\sigma,s}(C_\alpha) \cdot \sigma(\alpha)(\bot)$, where $\sigma(\alpha)(\bot)$ is the probability of chosing no transitions, i.e., of terminating the computation after α has occurred.

We say that there is a *weak combined transition* from $s \in S$ to $\mu \in \text{Disc}(S)$ labeled by $a \in \Sigma$ that is induced by σ, denoted by $s \overset{a}{\Longrightarrow}_{\text{c}} \mu$, if there exists a scheduler σ such that the following holds for the induced probabilistic execution fragment $\mu_{\sigma,s}$: (1) $\mu_{\sigma,s}(frags^*(\mathcal{A})) = 1$; (2) for each $\alpha \in frags^*(\mathcal{A})$, if $\mu_{\sigma,s}(\alpha) > 0$ then $trace(\alpha) = trace(a)$; (3) for each state t, $\mu_{\sigma,s}(\{\, \alpha \in frags^*(\mathcal{A}) \mid last(\alpha) = t \,\}) = \mu(t)$. See [23] for more details on weak combined transitions.

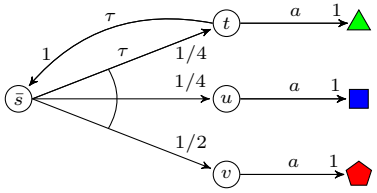


Figure 1 The PA $\mathcal{E}$

▶ **Example 1.** Consider the automaton $\mathcal{E}$ depicted in Figure 1 and denote by tr the only transition enabled by $\bar{s}$; $\mathcal{E}$ enables the weak combined transition $\bar{s} \overset{a}{\Longrightarrow}_{\text{c}} \mu$ where $\mu = \{\frac{1}{16}▲, \frac{5}{16}■, \frac{10}{16}⬟\}$ via the scheduler σ defined as follows: $\sigma(\bar{s}) = \sigma(\bar{s}\tau t \tau \bar{s}) = \delta_{tr}$, $\sigma(\bar{s}\tau t) = \delta_{t \xrightarrow{\tau} \delta_{\bar{s}}}$, $\sigma(\bar{s}\tau u) = \sigma(\bar{s}\tau t \tau \bar{s} \tau u) = \delta_{u \xrightarrow{a} \delta_■}$, $\sigma(\bar{s}\tau v) = \sigma(\bar{s}\tau t \tau \bar{s} \tau v) = \delta_{v \xrightarrow{a} \delta_⬟}$, $\sigma(\bar{s}\tau t \tau \bar{s} \tau t) = \delta_{t \xrightarrow{a} \delta_▲}$, and $\sigma(\alpha) = \delta_\bot$ for each other $\alpha \in frags^*(\mathcal{E})$. For instance, state ■ is reached with probability $\mu_{\sigma,\bar{s}}(\{\, \alpha \in frags^*(\mathcal{E}) \mid last(\alpha) = ■ \,\}) = \mu_{\sigma,\bar{s}}(\{\bar{s}\tau u a■, \bar{s}\tau t \tau \bar{s} \tau u a■\}) = 1{\cdot}1{\cdot}\frac{1}{4}{\cdot}1{\cdot}1{\cdot}1 + 1{\cdot}1{\cdot}\frac{1}{4}{\cdot}1{\cdot}1{\cdot}1{\cdot}\frac{1}{4}{\cdot}1{\cdot}1{\cdot}1 = \frac{5}{16} = \mu(■)$, as required. As we will see in Sect. 3.4, the same weak combined transition is induced also by a determinate scheduler.

We say that there is a *hyper-transition* from $\rho \in \text{Disc}(S)$ to $\mu \in \text{Disc}(S)$ labeled by $a \in \Sigma$, denoted by $\rho \overset{a}{\Longrightarrow}_{\text{c}} \mu$, if there exists a family of weak combined transitions $\{s \overset{a}{\Longrightarrow}_{\text{c}} \mu_s\}_{s \in \text{Supp}(\rho)}$ such that $\mu = \sum_{s \in \text{Supp}(\rho)} \rho(s) \cdot \mu_s$, i.e., for each $t \in S$, $\mu(t) = \sum_{s \in \text{Supp}(\rho)} \rho(s) \cdot \mu_s(t)$.

▶ **Definition 1.** Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two probabilistic automata. An equivalence relation $\mathcal{R}$ on the disjoint union $S_1 \uplus S_2$ is a *weak probabilistic bisimulation* if, for each pair of states $s, t \in S_1 \uplus S_2$ such that $s \mathcal{R} t$, if $s \xrightarrow{a} \mu_s$ for some probability distribution μ_s, then there exists a probability distribution μ_t such that $t \overset{a}{\Longrightarrow}_{\text{c}} \mu_t$ and $\mu_s \, \mathcal{L}(\mathcal{R}) \, \mu_t$.

Two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are weakly probabilistic bisimilar if there exists a weak probabilistic bisimulation $\mathcal{R}$ on $S_1 \uplus S_2$ such that $\bar{s}_1 \mathcal{R} \bar{s}_2$. We denote the coarsest

weak probabilistic bisimulation by $\approx$, and call it weak probabilistic bisimilarity.

This is the central definition around which the paper revolves. Weak probabilistic bisimilarity is an equivalence relation preserved by standard process algebraic composition operators on *PA* [20]. The complexity of deciding $\mathcal{A}_1 \approx \mathcal{A}_2$ strictly depends on finding $t \stackrel{a}{\Longrightarrow}_c \mu_t$ for which determinate schedulers suffice (cf. [3, Prop. 1]): in Section 3 we will show how to find them in polynomial time. The definition of bisimulation can be reformulated as follows, by simple manipulation of quantifiers:

▶ **Definition 2.** Given two *PA*s $\mathcal{A}_1$, $\mathcal{A}_2$, an equivalence relation $\mathcal{R}$ on $S_1 \uplus S_2$ is a *weak probabilistic bisimulation* if, for each transition $(s, a, \mu_s) \in D_1 \uplus D_2$ and each state t such that $s \mathcal{R} t$, there exists μ_t such that $t \stackrel{a}{\Longrightarrow}_c \mu_t$ and $\mu_s \mathcal{L}(\mathcal{R}) \mu_t$.

3 Weak Transition Construction as a Linear Programming Problem

We now discuss key elements of a decision algorithm for weak probabilistic bisimilarity. As we will see in Section 4, the core ingredient - and the source of the exponential complexity of the decision algorithm of [3] - is the recurring need to verify the step condition, that is, given a challenging transition $s \xrightarrow{a} \mu$ and $(s, t) \in \mathcal{R}$, to check whether there exists $t \stackrel{a}{\Longrightarrow}_c \mu_t$ such that $\mu \mathcal{L}(\mathcal{R}) \mu_t$.

With some inspiration from network flow problems, we will be able to see a transition $t \stackrel{a}{\Longrightarrow}_c \mu_t$ of the *PA* $\mathcal{A}$ as a *flow* where the initial probability mass δ_t flows and splits along internal transitions (and exactly one transition with label a for each stream when $a \neq \tau$) accordingly to the transition target distributions and the resolution of the nondeterminism performed by the scheduler.

This will allow us to arrive at a polynomial time algorithm to verify or refute the existence of a weak combined transition $t \stackrel{a}{\Longrightarrow}_c \mu_t$ such that $\mu \mathcal{L}(\mathcal{R}) \mu_t$.

3.1 Allowed Transitions

For the construction we are going to develop, we consider a more general case where we parametrize the scheduler so as to choose only specific, allowed, transitions when resolving the nondeterministic choices in a weak combined transition. This generalization will later be exploited by enabling us to generate tailored and thereby smaller LP-problems. For the intuition of this generalization, consider, for example, an automaton $\mathcal{C}$ that models a communication channel: it receives the information to transmit from the sender through an external action, then it performs an internal transition to represent the sending of the message on the communication channel, and finally it sends the transmitted information to the receiver. The communication channel is chosen nondeterministically between a reliable channel and an acknowledged lossy channel. If we want to check whether $\mathcal{C}$ always ensures the correct transmission of the received information, we can restrict the scheduler to choose only the lossy channel, i.e., we *allow* only the transitions relative to the lossy channel; if we impose this restriction and $\mathcal{C}$ is able to send eventually the transmitted information to the receiver with probability 1, then we can say that $\mathcal{C}$ always ensures the correct transmission of the received information.

▶ **Definition 3** (Allowed weak combined transition)**.** Given a *PA* $\mathcal{A}$ and a set of *allowed* transitions $A \subseteq D$, we say that there is an *allowed weak combined transition* from s to μ with label a respecting A, denoted by $s \stackrel{a}{\Longrightarrow}_c^A \mu$, if there exists a scheduler σ that induces $s \stackrel{a}{\Longrightarrow}_c \mu$ such that for each $\alpha \in frags^*(\mathcal{A})$, $\mathrm{Supp}(\sigma(\alpha)) \subseteq A$.

It is immediate to see that, when we consider every transition as allowed, i.e., $A = D$, the allowed weak combined transition $s \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \mu$ is just the usual transition $s \stackrel{a}{\Longrightarrow}_{\mathrm{c}} \mu$.

▶ **Proposition 4.** *Given a PA $\mathcal{A}$, a state s, and action a, and a probability distribution $\mu \in \mathrm{Disc}(S)$, there exists a scheduler σ_D for $\mathcal{A}$ that induces $s \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \mu$ if and only if there exists a scheduler σ for $\mathcal{A}$ that induces $s \stackrel{a}{\Longrightarrow}_{\mathrm{c}} \mu$.*

Similarly, we say that there is an *allowed hyper-transition* from a distribution over states ρ to a distribution over states μ labeled by a respecting A, denoted by $\rho \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu$, if there exists a family of allowed weak combined transitions $\{s \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu_s\}_{s \in \mathrm{Supp}(\rho)}$ such that $\mu = \sum \rho(s) \cdot \mu_s$.

An equivalent definition of allowed hyper-transition $\rho \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu$ is the following: given a *PA* $\mathcal{A}$, we say that there is an *allowed hyper-transition* from a distribution over states ρ to a distribution over states μ labeled by a respecting A if there exists an allowed weak combined transition $h \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A_h} \mu$ for the *PA* $\mathcal{A}_h = (S \cup \{h\}, \bar{s}, \Sigma, D \cup \{h \xrightarrow{\tau} \rho\})$ where $h \notin S$ and $A_h = A \cup \{h \xrightarrow{\tau} \rho\}$.

▶ **Proposition 5.** *Given a PA $\mathcal{A}$, $h \notin S$, $a \in \Sigma$, $A \subseteq D$, and $\rho, \mu \in \mathrm{Disc}(S)$, let $\mathcal{A}_h$ be the PA $\mathcal{A}_h = (S \cup \{h\}, \bar{s}, \Sigma, D \cup \{h \xrightarrow{\tau} \rho\})$ and A_h be $A \cup \{h \xrightarrow{\tau} \rho\}$.*

$\rho \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu$ exists in $\mathcal{A}$ if and only if $h \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A_h} \mu$ exists in $\mathcal{A}_h$.

▶ **Example 1 (cont.).** If we consider again the automaton $\mathcal{E}$ in Figure 1 and the set of allowed transitions $A = D \setminus \{t \xrightarrow{\tau} \delta_{\bar{s}}\}$, it is immediate to see that $\bar{s} \stackrel{a}{\Longrightarrow}_{\mathrm{c}} \mu$ with $\mu = \{\frac{1}{16}\blacktriangle, \frac{5}{16}\blacksquare, \frac{10}{16}\pentagon\}$ is not an allowed weak combined transition respecting A and that the only allowed weak combined transition with label a enabled by $\bar{s}$ is $\bar{s} \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \rho$ having $\rho = \{\frac{1}{4}\blacktriangle, \frac{1}{4}\blacksquare, \frac{1}{2}\pentagon\}$ as target distribution.

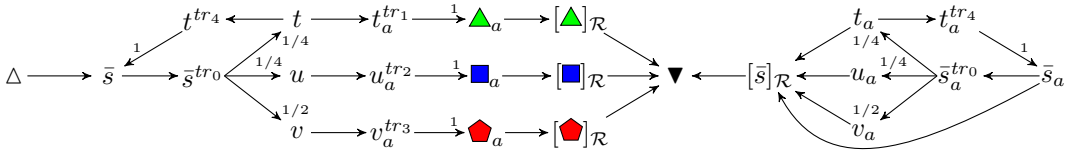
3.2 A Linear Programming Problem

We now assume we are given the *PA* $\mathcal{A}$, the set of allowed transitions $A \subseteq D$, the state t, the action a, the probability distribution μ, and the equivalence relation $\mathcal{R}$ on S. We intend to verify or refute the existence of a weak combined transition $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu_t$ of $\mathcal{A}$ satisfying $\mu \, \mathcal{L}(\mathcal{R}) \, \mu_t$ via the construction of a flow through the network graph $G(t, a, \mu, A, \mathcal{R}) = (V, E)$ defined as follows: for $a \neq \tau$, the set of vertices is $V = \{\triangle, \blacktriangledown\} \cup S \cup S^{tr} \cup S_a \cup S_a^{tr} \cup (S/\mathcal{R})$ where $S^{tr} = \{\, v^{tr} \mid tr = v \xrightarrow{b} \rho \in A, b \in \{a, \tau\} \,\}$, $S_a = \{\, v_a \mid v \in S \,\}$ and $S_a^{tr} = \{\, v_a^{tr} \mid v^{tr} \in S^{tr} \,\}$ and the set of arcs is $E = \{(\triangle, t)\} \cup \{\, (v_a, \mathcal{C}), (\mathcal{C}, \blacktriangledown) \mid \mathcal{C} \in S/\mathcal{R}, v \in \mathcal{C} \,\} \cup \{\, (v, v^{tr}), (v^{tr}, v'), (v_a, v_a^{tr}), (v_a^{tr}, v_a') \mid tr = v \xrightarrow{\tau} \rho \in A, v' \in \mathrm{Supp}(\rho) \,\} \cup \{\, (v, v_a^{tr}), (v_a^{tr}, v_a') \mid tr = v \xrightarrow{a} \rho \in A, v' \in \mathrm{Supp}(\rho) \,\}$. For $a = \tau$ the definition is similar: $V = \{\triangle, \blacktriangledown\} \cup S \cup S^{tr} \cup (S/\mathcal{R})$ and $E = \{(\triangle, t)\} \cup \{\, (v, \mathcal{C}), (\mathcal{C}, \blacktriangledown) \mid \mathcal{C} \in S/\mathcal{R}, v \in \mathcal{C} \,\} \cup \{\, (v, v^{tr}), (v^{tr}, v') \mid tr = v \xrightarrow{\tau} \rho \in A, v' \in \mathrm{Supp}(\rho) \,\}$.

$\triangle$ and $\blacktriangledown$ are two vertices that represent the source and the sink of the network, respectively. The graph encodes possible sequences of internal transitions, keeping track of which transition has happened by means of the vertices superscripted with tr, for this the set S^{tr} contains vertices that model the transitions of the automaton. The subsets of vertices subscripted by a are used to record that action a has happened already. Notably, not every vertex is used for defining arcs: the vertices v^{tr} where $tr = v \xrightarrow{b} \rho \in A$ and $b = a \neq \tau$ are used only to define the corresponding vertices v_a^{tr} that are actually involved in the definition of the set E of arcs. We could have removed these vertices from S^{tr} but this reduces the readability of the definition of S_a^{tr} without giving us a valuable effect on the computational complexity of the proposed solution.

▶ **Example 1 (cont.).** Consider the automaton $\mathcal{E}$ in Figure 1 and suppose that we want to check whether there exists an allowed weak combined transition $\bar{s} \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \rho$ such that $\rho \, \mathcal{L}(\mathcal{R}) \, \mu$

where $\mu = \{\frac{1}{16}\blacktriangle, \frac{5}{16}\blacksquare, \frac{10}{16}\pentagon\}$ and the classes induced by $\mathcal{R}$ are $\{\{\bar{s}, t, u, v\}, \{\blacktriangle\}, \{\blacksquare\}, \{\pentagon\}\}$. Let $tr_0 = \bar{s} \xrightarrow{\tau} \{\frac{1}{4}t, \frac{1}{4}u, \frac{1}{2}v\}$, $tr_1 = t \xrightarrow{a} \delta_{\blacktriangle}$, $tr_2 = u \xrightarrow{a} \delta_{\blacksquare}$, $tr_3 = v \xrightarrow{a} \delta_{\pentagon}$, and $tr_4 = t \xrightarrow{\tau} \delta_{\bar{s}}$. The network $G(\bar{s}, a, \mu, D, \mathcal{R})$ is as follows, where we omit vertices $\blacktriangle$, $\blacksquare$, and $\pentagon$ since they are not involved in any arc. Numbers attached to arcs indicate probabilities, and are not part of the graph.



Our intention is to use the network $G(t, a, \mu, A, \mathcal{R})$, in a maximum flow problem, since solving the latter has polynomial complexity. Unfortunately, the resulting problem does not model an allowed weak combined transition because probabilities are as such not necessarily respected: in ordinary flow problems we can not enforce a proportional balancing between the flows out of a given vertex. Instead, the entire incoming flow might be sent over a single outgoing arc, provided that the arc capacity is respected, while zero flow is sent over other arcs. In particular, we have no way to force the flow to split proportionally to the target probability distribution of a transition when the flow is less than 1. Apart from that, there is no obvious way to assign arc capacities since imposing capacity 1 to arcs is not always correct even if this is the maximum value for a probability. This problem is specifically caused by cycles of internal transitions. For self loops like $s \xrightarrow{\tau} \rho$ with $\rho(s) > 0$, one might after some reflection come up with a capacity $1/(1-p)$ where $p = \rho(s)$, but this does not extend to arbitrary τ-connected components.

For these reasons, we have to proceed differently: since any maximum flow problem can be expressed as a Linear Programming (LP) problem, we follow this path, but then refine the LP problem further, in order to eventually define a maximization problem whose solution is indeed equivalent to an allowed weak combined transition, as we will show in Section 3.4. For this, we use the above transformation of the automaton into a network graph as the starting point for generating an LP problem, which is afterwards enriched with additional constraints: we adopt the same notation of the max flow problem so we use $f_{u,v}$ to denote the "flow" through the arc from u to v. The *balancing factor* is a new concept we introduce to model a probabilistic choice and to ensure a balancing between flows that leave a vertex representing a probabilistic choice, i.e., leaving a vertex $v \in S^{tr} \cup S^{tr}_a$.

▶ **Definition 6** (The $t \stackrel{a}{\Longrightarrow}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ LP problem)**.** For $a \neq \tau$ we define the LP problem $t \stackrel{a}{\Longrightarrow}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ associated to the network graph $(V, E) = G(t, a, \mu, A, \mathcal{R})$ as follows.

$\max \sum_{(x,y)\in E} -f_{x,y}$

subject to

$f_{u,v} \geq 0$	for each $(u, v) \in E$
$f_{\triangle,t} = 1$	
$f_{\mathcal{C},\blacktriangledown} = \mu(\mathcal{C})$	for each $\mathcal{C} \in S/\mathcal{R}$
$\sum_{u\in\{\,x\mid(x,v)\in E\,\}} f_{u,v} - \sum_{u\in\{\,y\mid(v,y)\in E\,\}} f_{v,u} = 0$	for each $v \in V \setminus \{\triangle, \blacktriangledown\}$
$f_{v^{tr},v'} - \rho(v') \cdot f_{v,v^{tr}} = 0$	for each $tr = v \xrightarrow{\tau} \rho \in A$ and $v' \in \mathrm{Supp}(\rho)$
$f_{v^{tr}_a,v'_a} - \rho(v') \cdot f_{v_a,v^{tr}_a} = 0$	for each $tr = v \xrightarrow{\tau} \rho \in A$ and $v' \in \mathrm{Supp}(\rho)$
$f_{v^{tr}_a,v'_a} - \rho(v') \cdot f_{v,v^{tr}_a} = 0$	for each $tr = v \xrightarrow{a} \rho \in A$ and $v' \in \mathrm{Supp}(\rho)$

When a is τ, the LP problem $t \stackrel{\tau}{\Longrightarrow}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ associated to $G(t, \tau, \mu, A, \mathcal{R})$ is defined as above without the last two groups of constraints. Note that $t \stackrel{a}{\Longrightarrow}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ constraints define a system of linear equations extended with the non-negativity of variables $f_{u,v}$ and this rules out solutions where some variable $f_{x,y}$ has an infinite value. Moreover this may be

used to improve the actual implementation of the solver. It is worthwhile to point out that the objective function $\max \sum_{(x,y)\in E} -f_{x,y}$ is actually equivalent to $\min \sum_{(x,y)\in E} f_{x,y}$, i.e., a weak transition can also be seen as a minimum cost flow problem plus balancing constraints.

We can define the objective function in several ways but this does not affect the equivalence of $t \stackrel{a}{\Longrightarrow}{}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ and allowed weak combined transitions: in fact, the equivalence is based on variables $f_{v_a,[v]_\mathcal{R}}$ and $f_{\mathcal{C},\blacktriangledown}$ (where $v \in S$ and $\mathcal{C} \in S/\mathcal{R}$) that represent the probability to reach each state v (and then stopping) and each equivalence class $\mathcal{C}$, respectively; by definition of $t \stackrel{a}{\Longrightarrow}{}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ we have that $\sum_{v\in\mathcal{C}} f_{v_a,\mathcal{C}} = f_{\mathcal{C},\blacktriangledown}$ and $f_{\mathcal{C},\blacktriangledown} = \mu(\mathcal{C})$, thus their value does not strictly depend on the objective function. When $a = \tau$, we have the same result, just replacing v_a with v.

The objective function we use allows us to rule out trivial self-loops: suppose that there exists a transition $tr = x \xrightarrow{\tau} \delta_x \in A$ that we model by arcs (x, x^{tr}) and (x^{tr}, x). The balancing constraint for such arcs is $f_{x^{tr},x} - 1 \cdot f_{x,x^{tr}} = 0$ that is satisfied for each value of $f_{x^{tr},x} = f_{x,x^{tr}}$; however, the maximum for the objective function can be reached only when $f_{x,x^{tr}} = 0$, that is, the self-loop is not used. Similarly, we obtain that the value of the flow involving vertices that can not be reached from the vertex t is null as well as when such vertices may be reached from t but the solution of the problem requires that the flow from the vertex t to them is null.

▶ **Example 1 (cont.).** Consider again the automaton $\mathcal{E}$ in Figure 1 and suppose that we want to check whether there exists an allowed weak combined transition $\bar{s} \stackrel{a}{\Longrightarrow}{}^{D}_{\mathrm{c}} \rho$ such that $\rho\ \mathcal{L}(\mathcal{R})\ \mu$ where $\mu = \{\frac{1}{16}\blacktriangle, \frac{5}{16}\blacksquare, \frac{10}{16}\pentagon\}$ and the only non-singleton class of $\mathcal{R}$ is $\{\bar{s}, t, u, v\}$. Let $tr_0 = \bar{s} \xrightarrow{\tau} \{\frac{1}{4}t, \frac{1}{4}u, \frac{1}{2}v\}$, $tr_1 = t \xrightarrow{a} \delta_{\blacktriangle}$, $tr_2 = u \xrightarrow{a} \delta_{\blacksquare}$, $tr_3 = v \xrightarrow{a} \delta_{\pentagon}$, and $tr_4 = t \xrightarrow{\tau} \delta_{\bar{s}}$.

Besides other constraints, the LP problem $\bar{s} \stackrel{a}{\Longrightarrow}{}^{D}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ has the following constraints:

$$
\begin{array}{lll}
f_{\triangle,\bar{s}} = 1 & f_{[\blacktriangle]_\mathcal{R},\blacktriangledown} = 1/16 & f_{[\blacksquare]_\mathcal{R},\blacktriangledown} = 5/16 \\
f_{[\pentagon]_\mathcal{R},\blacktriangledown} = 10/16 & f_{\bar{s},\bar{s}^{tr_0}} - f_{\bar{s}^{tr_0},t} - f_{\bar{s}^{tr_0},u} - f_{\bar{s}^{tr_0},v} = 0 & f_{\triangle,\bar{s}} + f_{t^{tr_4},\bar{s}} - f_{\bar{s},\bar{s}^{tr_0}} = 0 \\
f_{\bar{s}^{tr_0},t} - f_{t,t_a^{tr_1}} - f_{t,t^{tr_4}} = 0 & f_{\bar{s}^{tr_0},u} - f_{u,u_a^{tr_2}} = 0 & f_{\bar{s}^{tr_0},v} - f_{v,v_a^{tr_3}} = 0 \\
f_{t,t_a^{tr_1}} - f_{t_a^{tr_1},\blacktriangle_a} = 0 & f_{u,u_a^{tr_2}} - f_{u_a^{tr_2},\blacksquare_a} = 0 & f_{v,v_a^{tr_3}} - f_{v_a^{tr_3},\pentagon_a} = 0 \\
f_{t,t^{tr_4}} - f_{t^{tr_4},\bar{s}} = 0 & f_{t_a^{tr_1},\blacktriangle_a} - f_{\blacktriangle_a,[\blacktriangle]_\mathcal{R}} = 0 & f_{u_a^{tr_2},\blacksquare_a} - f_{\blacksquare_a,[\blacksquare]_\mathcal{R}} = 0 \\
f_{v_a^{tr_3},\pentagon_a} - f_{\pentagon_a,[\pentagon]_\mathcal{R}} = 0 & f_{\blacktriangle_a,[\blacktriangle]_\mathcal{R}} - f_{[\blacktriangle]_\mathcal{R},\blacktriangledown} = 0 & f_{\blacksquare_a,[\blacksquare]_\mathcal{R}} - f_{[\blacksquare]_\mathcal{R},\blacktriangledown} = 0 \\
f_{\pentagon_a,[\pentagon]_\mathcal{R}} - f_{[\pentagon]_\mathcal{R},\blacktriangledown} = 0 & f_{\bar{s}^{tr_0},t} - 1/4 f_{\bar{s},\bar{s}^{tr_0}} = 0 & f_{\bar{s}^{tr_0},u} - 1/4 f_{\bar{s},\bar{s}^{tr_0}} = 0 \\
f_{\bar{s}^{tr_0},v} - 1/2 f_{\bar{s},\bar{s}^{tr_0}} = 0 & f_{t_a^{tr_1},\blacktriangle_a} - 1 f_{t,t_a^{tr_1}} = 0 & f_{u_a^{tr_2},\blacksquare_a} - 1 f_{u,u_a^{tr_2}} = 0 \\
f_{v_a^{tr_3},\pentagon_a} - 1 f_{v,v_a^{tr_3}} = 0 & f_{t^{tr_4},\bar{s}} - 1 f_{t,t^{tr_4}} = 0 &
\end{array}
$$

A solution that maximizes the objective function sets all variables to value 0 except for

$$
\begin{array}{llllllll}
f_{\triangle,\bar{s}} & = 16/16 & f_{[\blacktriangle]_\mathcal{R},\blacktriangledown} & = 1/16 & f_{[\blacksquare]_\mathcal{R},\blacktriangledown} & = 5/16 & f_{[\pentagon]_\mathcal{R},\blacktriangledown} & = 10/16 \\
f_{\bar{s},\bar{s}^{tr_0}} & = 20/16 & f_{\bar{s}^{tr_0},t} & = 5/16 & f_{\bar{s}^{tr_0},u} & = 5/16 & f_{\bar{s}^{tr_0},v} & = 10/16 \\
f_{t,t_a^{tr_1}} & = 1/16 & f_{t,t^{tr_4}} & = 4/16 & f_{u,u_a^{tr_2}} & = 5/16 & f_{v,v_a^{tr_3}} & = 10/16 \\
f_{t_a^{tr_1},\blacktriangle_a} & = 1/16 & f_{t^{tr_4},\bar{s}} & = 4/16 & f_{u_a^{tr_2},\blacksquare_a} & = 5/16 & f_{v_a^{tr_3},\pentagon_a} & = 10/16 \\
f_{\blacktriangle_a,[\blacktriangle]_\mathcal{R}} & = 1/16 & f_{\blacksquare_a,[\blacksquare]_\mathcal{R}} & = 5/16 & f_{\pentagon_a,[\pentagon]_\mathcal{R}} & = 10/16 & &
\end{array}
$$

The variable $f_{\bar{s},\bar{s}^{tr_0}} = 20/16$ is part of a cycle and its value is greater than 1, confirming that 1, the maximum probability, in general is not a proper value for arc capacities.

3.3 Complexity of the LP Problem

We analyze the complexity of $t \stackrel{a}{\Longrightarrow}{}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ for $a \neq \tau$ since $t \stackrel{\tau}{\Longrightarrow}{}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$ is just a special case of $t \stackrel{a}{\Longrightarrow}{}^{A}_{\mathrm{c}} \diamond \mathcal{L}(\mathcal{R})\ \mu$.

Given the automaton $\mathcal{A}$ and the set $A \subseteq D$ of allowed transitions, let $N_S = |S|$, $N_A = |A|$, and $N = \max\{N_S, N_A\}$. Suppose that $a \neq \tau$ and consider the network graph $G(t, a, \mu, A, \mathcal{R}) = (V, E)$. The cardinality of V is: $|V| \leq 2+N_S+N_A+N_S+N_A+N_S \in \mathcal{O}(N)$ and the cardinality of E is: $|E| \leq 1+2N_S+2(N_S+1)N_A+(N_S+1)N_A \in \mathcal{O}(N^2)$. Note that this is also the cost of generating the $G(t, a, \mu, A, \mathcal{R})$ network graph from the automaton $\mathcal{A}$.

Now, consider the $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ LP problem: the number of variables is $|\{\, f_{u,v} \mid (u, v) \in E \,\}| = |E| \in \mathcal{O}(N^2)$ and the number of constraints is $|E| + 1 + N_S + N_S N_A + N_S N_A + N_S N_A + |V| - 2 \in \mathcal{O}(N^2)$, so generating $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ is polynomial in N. Since there exist polynomial algorithms for solving LP problems [26], solving the $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ problem is polynomial in N. The implementation of $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ can be optimized in several ways (cf. [15, Sec. 3.4]) without changing the resulting complexity class.

▶ **Theorem 7.** *Given a PA $\mathcal{A}$, an equivalence relation $\mathcal{R}$ on S, an action a, a probability distribution $\mu \in \mathrm{Disc}(S)$, a set of allowed transitions $A \subseteq D$, and a state $t \in S$, consider the problem $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ as defined above. Let $N = \max\{|S|, |A|\}$.*

Generating and checking the existence of a valid solution of the $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ LP problem is polynomial in N.

3.4 Equivalence of LP Problems and Weak Transitions

In this section we present the main theorem that equates $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ with an allowed weak combined transition, that is, $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ has a solution if and only if there exists a scheduler σ for $\mathcal{A}$ that induces $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu_t$ such that $\mu\ \mathcal{L}(\mathcal{R})\ \mu_t$. This result easily extends to ordinary weak combined transitions and hyper-transitions.

▶ **Theorem 8.** *Given a PA $\mathcal{A}$, an equivalence relation $\mathcal{R}$ on S, an action a, a probability distribution $\mu \in \mathrm{Disc}(S)$, a set of allowed transitions $A \subseteq D$, and a state $t \in S$, consider the problem $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ as defined above.*

$t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \circledast \mathcal{L}(\mathcal{R})\ \mu$ has a solution f^* such that $f^*_{\mathcal{C},\blacktriangledown} = \mu(\mathcal{C})$ for each $\mathcal{C} \in S/\mathcal{R}$ if and only if there exists a scheduler σ for $\mathcal{A}$ that induces $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu_t$ such that $\mu\ \mathcal{L}(\mathcal{R})\ \mu_t$.

Proof outline. The scheduler σ we define in the proof for the "only if" direction assigns to each execution fragment α with $last(\alpha) = v$ the sub-probability distribution over transitions defined, for each transition $tr \in A$ such that $src(tr) = v$, as the ratio $f^*_{v_{\mathtt{t}}, v^{tr}_{\bar{\mathtt{t}}}} / \vec{f}^*_{v_{\mathtt{t}}}$, given that $\vec{f}^*_{v_{\mathtt{t}}} > 0$, where $\vec{f}^*_v$ is the total flow incoming v, $\mathtt{t} = trace(\alpha)$, and $\bar{\mathtt{t}}$ is the concatenation of $trace(\alpha)$ and of $trace(act(tr))$. The remaining probability of stopping in the state v is exactly $f^*_{v_{\mathtt{t}},[v]_{\mathcal{R}}} / \vec{f}^*_{v_{\mathtt{t}}}$. The way we generate the network $G(t, a, \mu, A, \mathcal{R})$ ensures that $f^*_{v_{\mathtt{t}}, v^{tr}_{\mathtt{t}}} = 0$ when $\mathtt{t} \notin \{\varepsilon, trace(a)\}$ and that $f^*_{v_{\mathtt{t}},[v]_{\mathcal{R}}} / \vec{f}^*_{v_{\mathtt{t}}} = 0$ when $\mathtt{t} \neq trace(a)$. The proof for the "if" direction is the dual, that is, we define a feasible solution f^* as the sum of the probabilities of the cones of execution fragments, i.e., $\bar{f}^*_{v_b} = \sum_{\alpha \in \{\, \phi \in frags^*(\mathcal{A}) \mid trace(\phi) = b \wedge last(\phi) = v \,\}} \mu_{\sigma,t}(C_\alpha)$; then the existence of such feasible solution is enough to prove that there exists a (possibly different) solution f^o that maximizes the objective function while preserving the property that for each $\mathcal{C} \in S/\mathcal{R}$, $f^o_{\mathcal{C},\blacktriangledown} = \mu(\mathcal{C})$. ◀

It is worth to observe that the resulting scheduler is a determinate scheduler and an immediate corollary of this theorem confirming and improving Proposition 3 of [3] is that each scheduler inducing $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu_t$ can be replaced by a determinate scheduler inducing $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu_t$ as well.

▶ **Example 1 (cont.).** We already know from the first part of this example that there exists a non-determinate scheduler σ inducing $\bar{s} \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \mu$ where $\mu = \{\frac{1}{16}\blacktriangle, \frac{5}{16}\blacksquare, \frac{10}{16}\pentagon\}$. So,

let again $tr_0 = \bar{s} \xrightarrow{\tau} \{\frac{1}{4}t, \frac{1}{4}u, \frac{1}{2}v\}$, $tr_1 = t \xrightarrow{a} \delta_{\blacktriangle}$, $tr_2 = u \xrightarrow{a} \delta_{\blacksquare}$, $tr_3 = v \xrightarrow{a} \delta_{\pentagon}$, and $tr_4 = t \xrightarrow{\tau} \delta_{\bar{s}}$. Theorem 8 ensures that there exists a scheduler σ', possibly different from σ, that induces $\bar{s} \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \mu$; in particular, σ' is the determinate scheduler defined as follows.

$$\sigma'(\alpha) = \begin{cases} \delta_{tr_0} & \text{if } \mathit{trace}(\alpha) = \varepsilon \text{ and } \mathit{last}(\alpha) = \bar{s}; \\ \{\frac{1}{5}tr_1, \frac{4}{5}tr_4\} & \text{if } \mathit{trace}(\alpha) = \varepsilon \text{ and } \mathit{last}(\alpha) = t; \\ \delta_{tr_2} & \text{if } \mathit{trace}(\alpha) = \varepsilon \text{ and } \mathit{last}(\alpha) = u; \\ \delta_{tr_3} & \text{if } \mathit{trace}(\alpha) = \varepsilon \text{ and } \mathit{last}(\alpha) = v; \\ \delta_{\bot} & \text{otherwise.} \end{cases}$$

It is straightforward to check that σ' actually induces $\bar{s} \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \mu$. For instance, state $\blacktriangle$ is reached with probability $\mu_{\sigma',\bar{s}}(\{\, \alpha \in \mathit{frags}^*(\mathcal{E}) \mid \mathit{last}(\alpha) = \blacktriangle \,\}) = \mu_{\sigma',\bar{s}}(\{\, \bar{s}\tau t(\tau\bar{s}\tau t)^n a\blacktriangle \mid n \in \mathbb{N} \,\}) = 1 \cdot \frac{1}{4} \cdot \sum_{n\in\mathbb{N}}(\frac{4}{5} \cdot 1 \cdot 1 \cdot \frac{1}{4})^n \cdot \frac{1}{5} \cdot 1 \cdot 1 = \frac{1}{4} \cdot \frac{1}{5} \cdot (1 - \frac{1}{5})^{-1} = \frac{1}{4} \cdot \frac{1}{5} \cdot \frac{5}{4} = \mu(\blacktriangle)$, as required.

▶ **Corollary 9.** *Given a PA $\mathcal{A}$, $t \in S$ and $h \notin S$, $a \in \Sigma$, $\rho, \mu, \mu_t \in \mathrm{Disc}(S)$, $A \subseteq D$, an equivalence relation $\mathcal{R}$ on S, a transition $h \xrightarrow{\tau} \rho$, $A_h = A \cup \{h \xrightarrow{\tau} \rho\}$, $D_h = D \cup \{h \xrightarrow{\tau} \rho\}$, and the PA $\mathcal{A}_h = (S \cup \{h\}, \bar{s}, \Sigma, D_h)$, the following holds:*

1. *$t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \diamond\, \mathcal{L}(\mathcal{R})\ \mu$ has a solution f^* such that $f^*_{\mathcal{C},\blacktriangledown} = \mu(\mathcal{C})$ for each $\mathcal{C} \in S/\mathcal{R}$ if and only if there exists a scheduler σ for $\mathcal{A}$ inducing $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ such that $\mu\ \mathcal{L}(\mathcal{R})\ \mu_t$;*
2. *$h \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A_h} \diamond\, \mathcal{L}(\mathcal{R})\ \mu$ ($h \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D_h} \diamond\, \mathcal{L}(\mathcal{R})\ \mu$) relative to $\mathcal{A}_h$ has a solution f^* such that $f^*_{\mathcal{C},\blacktriangledown} = \mu(\mathcal{C})$ for each $\mathcal{C} \in S/\mathcal{R}$ if and only if there exists a scheduler σ for $\mathcal{A}$ inducing $\rho \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \mu_t$ ($\rho \stackrel{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$, respectively) such that $\mu_t\ \mathcal{L}(\mathcal{R})\ \mu$.*

When $\mathcal{R}$ is the identity relation $\mathcal{I}$, $\mu\ \mathcal{L}(\mathcal{I})\ \mu_t$ implies $\mu_t = \mu$.

Proof outline. The corollary follows directly from a combination of Theorem 8 for the equivalence between the LP problem and allowed weak combined transition, Proposition 4 for weak combined transitions, and Proposition 5 for hyper-transitions. ◀

It is worth to note also that, when we consider an *MDP* as a *PA*, the $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \diamond\, \mathcal{L}(\mathcal{R})\ \mu$ construction allows us to solve the multi-objective reachability problem on *MDP*s, that is, given an *MDP* and a collection of disjoint target sets $E_1, \ldots, E_k$ and goal probabilities $p_1, \ldots, p_k$, check whether there exists a scheduler that reaches E_i with probability exactly p_i.

3.5 Equivalence Matching

Theorem 8 and its corollary allow us to check in polynomial time whether it is possible to reach a given probability distribution μ from a state t or a probability distribution ρ. We now consider a more general case where, given a *PA* $\mathcal{A}$, two distributions $\rho_1, \rho_2 \in \mathrm{Disc}(S)$, two actions $a_1, a_2 \in \Sigma$, two sets $A_1, A_2 \subseteq D$ of allowed transitions, and an equivalence relation $\mathcal{R}$ on S, we want to check in polynomial time whether there exist $\mu_1, \mu_2 \in \mathrm{Disc}(S)$ such that $\rho_1 \stackrel{a_1}{\Longrightarrow}_{\mathrm{c}}^{A_1} \mu_1$, $\rho_2 \stackrel{a_2}{\Longrightarrow}_{\mathrm{c}}^{A_2} \mu_2$, and $\mu_1\ \mathcal{L}(\mathcal{R})\ \mu_2$. In order to find μ_1 and μ_2, we can consider a family $\{p_{\mathcal{C}}\}_{\mathcal{C}\in S/\mathcal{R}}$ of non-negative values such that $\sum_{\mathcal{C}\in S/\mathcal{R}} p_{\mathcal{C}} = 1$ and a probability distribution $\bar{\mu}$ satisfying $\bar{\mu}(\mathcal{C}) = p_{\mathcal{C}}$ for each $\mathcal{C} \in S/\mathcal{R}$ and then solve $\rho_1 \stackrel{a_1}{\Longrightarrow}_{\mathrm{c}}^{A_1} \diamond\, \mathcal{L}(\mathcal{R})\ \bar{\mu}$ and $\rho_2 \stackrel{a_2}{\Longrightarrow}_{\mathrm{c}}^{A_2} \diamond\, \mathcal{L}(\mathcal{R})\ \bar{\mu}$ where $\rho \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A} \diamond\, \mathcal{L}(\mathcal{R})\ \mu$ is the problem $h \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{A_h} \diamond\, \mathcal{L}(\mathcal{R})\ \mu$ relative to $\mathcal{A}_h = (S \cup \{h\}, \bar{s}, \Sigma, D \cup \{h \xrightarrow{\tau} \rho\})$ with $h \notin S$ and $A_h = A \cup \{h \xrightarrow{\tau} \rho\}$. The main problem of this approach is to find a good family of values $p_{\mathcal{C}}$; since we do not care about actual values, we consider $p_{\mathcal{C}}$ as variables satisfying $p_{\mathcal{C}} \geq 0$ and $\sum_{\mathcal{C}\in S/\mathcal{R}} p_{\mathcal{C}} = 1$ and we define the LP problem $P_{1,2}$ derived from $P_1 = \rho_1 \stackrel{a_1}{\Longrightarrow}_{\mathrm{c}}^{A_1} \diamond\, \mathcal{L}(\mathcal{R})\ \bar{\mu}$ and $P_2 = \rho_2 \stackrel{a_2}{\Longrightarrow}_{\mathrm{c}}^{A_2} \diamond\, \mathcal{L}(\mathcal{R})\ \bar{\mu}$ as follows (after renaming of P_2 variables to avoid collisions): the objective function of $P_{1,2}$ is the sum

of the objective functions of P_1 and P_2; the set of constraints of $P_{1,2}$ is $\sum_{\mathcal{C}\in S/\mathcal{R}} p_{\mathcal{C}} = 1$ together with $p_{\mathcal{C}} \geq 0$ for $\mathcal{C} \in S/\mathcal{R}$ and the union of the sets of constraints of P_1 and P_2 where each occurrence of $\bar{\mu}(\mathcal{C})$ is replaced by $p_{\mathcal{C}}$.

It is quite easy to verify that $P_{1,2}$ has a solution if and only if both P_1 and P_2 have a solution (with respect to the same $\bar{\mu}$) and thus, by Corollary 9(2), if and only if ρ_1 and ρ_2 enable an allowed hyper-transition to μ_1 and μ_2, respectively, such that $\mu_1\ \mathcal{L}(\mathcal{R})\ \mu_2$, as required. It is immediate to see that $P_{1,2}$ can still be solved in polynomial time, since it is just the union of P_1 and P_2 extended with at most $|S|$ variables and $2|S|$ constraints.

▶ **Proposition 10.** *Given a PA $\mathcal{A}$, $\rho_1, \rho_2 \in \mathrm{Disc}(S)$, $a_1, a_2 \in \Sigma$, two sets $A_1, A_2 \subseteq D$ of allowed transitions, and an equivalence relation $\mathcal{R}$ on S, the existence of $\mu_1, \mu_2 \in \mathrm{Disc}(S)$ such that $\rho_1 \stackrel{a_1}{\Longrightarrow}_{\mathrm{c}}^{A_1} \mu_1$, $\rho_2 \stackrel{a_2}{\Longrightarrow}_{\mathrm{c}}^{A_2} \mu_2$, and $\mu_1\ \mathcal{L}(\mathcal{R})\ \mu_2$ can be checked in polynomial time.*

The above proposition easily extends, by Corollary 9, to each combination of weak combined transitions, allowed hyper-transitions, and allowed weak combined transitions as well as to exact matching as induced by the identity relation $\mathcal{I}$.

4 Decision Procedure

In this section, we recast the decision procedure of [3] that decides whether two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar according to $\approx$, that is, whether $\mathcal{A}_1 \approx \mathcal{A}_2$, following the standard partition refinement approach [3, 17, 19, 21]. More precisely, procedure QUOTIENT iteratively constructs the set $S/\!\approx$, the set of equivalence classes of states $S = S_1 \uplus S_2$ under $\approx$, starting with the partitioning $\mathcal{W} = \{S\}$ and refining it until $\mathcal{W}$ satisfies the definition of weak probabilistic bisimulation and thus the resulting partitioning is the coarsest one, i.e., we compute the weak probabilistic bisimilarity.

QUOTIENT$(\mathcal{A}_1, \mathcal{A}_2)$

$\mathcal{W} = \{S_1 \uplus S_2\}$;
$(\mathcal{C}, a, \mu) =$ FINDSPLIT$(\mathcal{W})$;
while $\mathcal{C} \neq \emptyset$ **do**
  $\mathcal{W} =$ REFINE$(\mathcal{W}, (\mathcal{C}, a, \mu))$;
  $(\mathcal{C}, a, \mu) =$ FINDSPLIT$(\mathcal{W})$;
return $\mathcal{W}$

Deciding whether two automata are bisimilar then reduces to check whether their start states belong to the same class. In the following, we treat $\mathcal{W}$ both as a set of partitions and as an equivalence relation without further mentioning.

FINDSPLIT$(\mathcal{W})$

1: **for all** $(s, a, \mu) \in D = D_1 \uplus D_2$ **do**
2: **for all** $t \in [s]_{\mathcal{W}}$ **do**
3: **if** $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \Diamond\, \mathcal{L}(\mathcal{W})\ \mu$ has no solution
4: **return** $([s]_{\mathcal{W}}, a, \mu)$
5: **return** $(\emptyset, \tau, \delta_{\bar{s}})$

The partitioning is refined by procedure REFINE into a finer partitioning as long as there is a partition containing two states that violate the bisimulation condition, which is checked for in procedure FINDSPLIT. Procedure REFINE, that we do not provide explicitly as in [3], splits partition $\mathcal{C}$ into two new partitions according to the discriminating information $(\mathcal{C}, a, \mu)$ identified by FINDSPLIT before. So far, the procedure is as the *DecideBisim*$(\mathcal{A}_1, \mathcal{A}_2)$ procedure proposed in [3].

The difference arises inside the procedure FINDSPLIT, where we check directly the step condition by solving for each transition $s \xrightarrow{a} \mu$ the LP problem $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \Diamond\, \mathcal{L}(\mathcal{W})\ \mu$ that has a solution, according to Corollary 9(1), if and only if there exists $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}} \mu_t$ such that $\mu\ \mathcal{L}(\mathcal{W})\ \mu_t$.

4.1 Complexity Analysis of the Procedure

Given two PAs $\mathcal{A}_1$ and $\mathcal{A}_2$, let $S = S_1 \uplus S_2$, $D = D_1 \uplus D_2$, and $N = \max\{|S|, |D|\}$.

In the worst case (that occurs when the current $\mathcal{W}$ satisfies the step condition), the **for** at line 1 of procedure FINDSPLIT is performed at most N times as well as the inner

for, so $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \diamond \mathcal{L}(\mathcal{W})\ \mu$ is generated and solved at most N^2 times. Since by Theorem 7 generating and checking the existence of a valid solution for $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \diamond \mathcal{L}(\mathcal{W})\ \mu$ is polynomial in N, this implies that also FindSplit is polynomial in N; more precisely, denoted by $p(N)$ the complexity of $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \diamond \mathcal{L}(\mathcal{W})\ \mu$, FindSplit $\in \mathcal{O}(N^2 p(N))$. Note that we can improve the running time required to solve the $t \stackrel{a}{\Longrightarrow}_{\mathrm{c}}^{D} \diamond \mathcal{L}(\mathcal{W})\ \mu$ LP problem by replacing D with D' at line 3 of FindSplit where D' contains only transitions with label τ or a enabled by states reachable from t.

The **while** loop in the procedure Quotient can be performed at most N times; this happens when in each loop the procedure FindSplit returns $(\mathcal{C}, a, \mu)$ where $\mathcal{C} \neq \emptyset$, that is, not every pair of states in $\mathcal{C}$ satisfies the step condition. Since in each loop the procedure Refine splits such class $\mathcal{C}$ in two classes $\mathcal{C}_1$ and $\mathcal{C}_2$, after at most N loops every class contains a single state and the procedure FindSplit returns $(\emptyset, \tau, \delta_{\bar{s}})$ since each transition $s \xrightarrow{a} \mu_s$ is obviously matched by s itself. Since Refine and FindSplit are polynomial in N, also Quotient is polynomial in N, thus checking $\mathcal{A}_1 \approx \mathcal{A}_2$ is polynomial in N.

▶ **Theorem 11.** *Given two PAs $\mathcal{A}_1$ and $\mathcal{A}_2$, let $N = \max\{|S_1 \uplus S_2|, |D_1 \uplus D_2|\}$. Checking $\mathcal{A}_1 \approx \mathcal{A}_2$ is polynomial in N.*

It is as yet open whether checking $\mathcal{A}_1 \approx \mathcal{A}_2$ can be done in strong-polynomial time, while this is known for strong bisimulation.

5 Concluding Remarks

This paper has established a polynomial time decision algorithm for *PA* weak probabilistic bisimulation, closing the quest for an effective decision algorithm coined in [3]. The core innovation is a novel characterization of weak combined transitions as an LP problem, enabling us to check the existence of a weak combined transition in polynomial time. The algorithm can be exploited in an effective compositional minimization strategy for *PA* (or *MDP*) and potentially also for Markov automata. Furthermore, the LP approach we developed is readily extensible to related problems requiring to find a specific weak transition. Another area of immediate applicability concerns cost-related problems where transition costs may relate to power or resource consumption in *PA* or *MDP*.

Acknowledgments. The authors are grateful to Christian Eisentraut (Saarland University) for insightful discussions. This work has been supported by the DFG as part of the SFB/TR 14 "Automatic Verification and Analysis of Complex Systems" (AVACS), by the DFG/NWO Bilateral Research Programme ROCKS, and by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS). Andrea Turrini has received support from the Cluster of Excellence "Multimodal Computing and Interaction" (MMCI), part of the German Excellence Initiative.

References

1 C. Baier and M. Stoelinga. Norm functions for probabilistic bisimulations with delays. In *FOSSACS*, vol. 1784 of *LNCS*, pages 1–16, 2000.

2 E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, J. Rakow, R. Wimmer, and B. Becker. Compositional dependability evaluation for STATEMATE. *IEEE Transactions on Software Engineering*, 35(2):274–292, 2009.

3 S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation. In *CONCUR*, vol. 2421 of *LNCS*, pages 371–385, 2002.

4 L. Cheung, M. Stoelinga, and F. W. Vaandrager. A testing scenario for probabilistic processes. *Journal of the ACM*, 54(6), 2007.

5 N. Coste, H. Hermanns, E. Lantreibecq, and W. Serwe. Towards performance prediction of compositional models in industrial GALS designs. In *CAV*, vol. 5643 of *LNCS*, pages 204–218, 2009.

6 Y. Deng and M. Hennessy. On the semantics of Markov automata. In *ICALP*, vol. 6756 of *LNCS*, pages 307–318, 2011.

7 C. Derman. *Finite State Markovian Decision Processes*. Academic Press, Inc., 1970.

8 C. Eisentraut, H. Hermanns, and L. Zhang. Concurrency and composition in a stochastic world. In *CONCUR*, vol. 6269 of *LNCS*, pages 21–39, 2010.

9 C. Eisentraut, H. Hermanns, and L. Zhang. On probabilistic automata in continuous time. In *LICS*, pages 342–351, 2010.

10 M.-A. Esteve, J.-P. Katoen, V. Y. Nguyen, B. Postma, and Y. Yushtein. Formal correctness, safety, dependability and performance analysis of a satellite. In *ICSE*, pages 1022–1031, 2012.

11 H. A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Department of Computer Systems, Uppsala University, 1991.

12 B. R. Haverkort, M. Kuntz, A. Remke, S. Roolvink, and M. Stoelinga. Evaluating repair strategies for a water-treatment facility using Arcade. In *DSN*, pages 419–424, 2010.

13 H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, vol. 2428 of *LNCS*. Springer Verlang, 2002.

14 H. Hermanns and J.-P. Katoen. Automated compositional Markov chain generation for a plain-old telephone system. *Science of Computer Programming*, 36(1):97–127, 2000.

15 H. Hermanns and A. Turrini. Deciding Probabilistic Automata weak bisimulation in polynomial time. Available at `http://arxiv.org/abs/1205.0376`.

16 A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS*, vol. 3920 of *LNCS*, pages 441–444, 2006.

17 P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

18 K. G. Larsen and A. Skou. Bisimulation through probabilistic testing (preliminary report). In *POPL*, pages 344–352, 1989.

19 R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

20 A. Parma and R. Segala. Axiomatization of trace semantics for stochastic nondeterministic processes. In QEST, pages 294–303, 2004.

21 A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In *CONCUR*, vol. 1877 of *LNCS*, pages 334–349, 2000.

22 R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.

23 R. Segala. Probability and nondeterminism in operational models of concurrency. In *CONCUR*, vol. 4137 of *LNCS*, pages 64–78, 2006.

24 R. Segala and A. Turrini. Comparative analysis of bisimulation relations on alternating and non-alternating probabilistic models. In *QEST*, pages 44–53, 2005.

25 M. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-Time and Parametric Systems*. PhD thesis, University of Nijmegen, the Netherlands, 2002.

26 M. J. Todd. The many facets of linear programming. *Math. Progr.*, 91(3):417–436, 2002.

27 M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Foundations of Computer Science*, pages 327–338, 1985.

28 R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, and B. Becker. Sigref - A symbolic bisimulation tool box. In *ATVA*, volume 4218 of LNCS, pages 477–492, 2006.

Bisimilarity of Probabilistic Pushdown Automata

Vojtěch Forejt[1], Petr Jančar[2], Stefan Kiefer[1], and James Worrell[1]

1 **Department of Computer Science, University of Oxford, UK**
2 **Department of Computer Science, FEI, Techn. Univ. Ostrava, Czech Republic**

Abstract

We study the bisimilarity problem for probabilistic pushdown automata (pPDA) and subclasses thereof. Our definition of pPDA allows both probabilistic and non-deterministic branching, generalising the classical notion of pushdown automata (without ε-transitions). Our first contribution is a general construction that reduces checking bisimilarity of probabilistic transition systems to checking bisimilarity of non-deterministic transition systems. This construction directly yields decidability of bisimilarity for pPDA, as well as an elementary upper bound for the bisimilarity problem on the subclass of probabilistic basic process algebras, i.e., single-state pPDA. We further show that, with careful analysis, the general reduction can be used to prove an EXPTIME upper bound for bisimilarity of probabilistic visibly pushdown automata. Here we also provide a matching lower bound, establishing EXPTIME-completeness. Finally we prove that deciding bisimilarity of probabilistic one-counter automata, another subclass of pPDA, is PSPACE-complete. Here we use a more specialised argument to obtain optimal complexity bounds.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases bisimilarity, probabilistic systems, pushdown automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.448

1 Introduction

Equivalence checking is the problem of determining whether two systems are semantically identical. This is an important question in automated verification and represents a line of research that can be traced back to the inception of theoretical computer science. A great deal of work in this area has been devoted to the complexity of *bisimilarity* for various classes of infinite-state systems related to grammars, such as one-counter automata, basic process algebras, and pushdown automata, see [4] for an overview. We mention in particular the landmark result showing the decidability of bisimilarity for pushdown automata [15].

In this paper we are concerned with probabilistic pushdown automata (pPDA), that is, pushdown automata with both non-deterministic and probabilistic branching. In particular, our pPDA generalise classical pushdown automata without ε-transitions. We refer to automata with only probabilistic branching as *fully probabilistic*.

We consider the complexity of checking bisimilarity for probabilistic pushdown automata and various subclasses thereof. The subclasses we consider are probabilistic versions of models that have been extensively studied in previous works [4, 16]. In particular, we consider probabilistic one-counter automata (pOCA), which are probabilistic pushdown automata with singleton stack alphabet; probabilistic Basic Process Algebras (pBPA), which are single-state probabilistic pushdown automata; probabilistic visibly pushdown automata (pvPDA), which are automata in which the stack action, whether to push or pop, for each transition is determined by the input letter. Probabilistic one-counter automata have been studied in

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 448–460

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

the classical theory of stochastic processes as *quasi-birth-death processes* [6]. Probabilistic BPA seems to have been introduced in [3].

While the complexity of bisimilarity for finite-state probabilistic automata is well understood [1, 5], there are relatively few works on equivalence of infinite-state probabilistic systems. Bisimilarity of probabilistic BPA was shown decidable in [3], but without any complexity bound. In [8] probabilistic simulation between probabilistic pushdown automata and finite state systems was studied.

1.1 Contribution

The starting point of the paper is a construction that can be used to reduce the bisimilarity problem for many classes of probabilistic systems to the bisimilarity problem for their non-probabilistic counterparts. The reduction relies on the observation that in the bisimilarity problem, the numbers that occur as probabilities in a probabilistic system can be "encoded" as actions in the non-probabilistic system. This comes at the price of an exponential blow-up in the branching size, but still allows us to establish several new results. It is perhaps surprising that there is a relatively simple reduction of probabilistic bisimilarity to ordinary bisimilarity. Hitherto it has been typical to establish decidability in the probabilistic case using bespoke proofs, see, e.g., [3, 8]. Instead, using our reduction, we can leverage the rich theory that has been developed in the non-probabilistic case.

The main results of the paper are as follows:

- Using the above-mentioned reduction together with the result of [15], we show that bisimilarity for probabilistic pushdown automata is decidable.
- For the subclass of probabilistic BPA, i.e., automata with a single control state, the same reduction yields a 3EXPTIME upper bound for checking bisimilarity via a doubly exponential procedure for bisimilarity on BPA [4] (see also [11]). This improves the result of [3], where only a decidability result was given without any complexity bound. An EXPTIME lower bound for this problem follows from the recent work of [12] for non-probabilistic systems.
- For probabilistic visibly pushdown automata, the above reduction immediately yields a 2EXPTIME upper bound. However we show that with more careful analysis we can extract an EXPTIME upper bound. In this case we also show EXPTIME-hardness, thus obtaining matching lower and upper bounds.
- For fully probabilistic one-counter automata we obtain matching lower and upper PSPACE bounds for the bisimilarity problem. In both cases the bounds are obtained by adapting constructions from the non-deterministic case [16, 2] rather than by using the generic reduction described above.

2 Preliminaries

Given a countable set A, a *probability distribution* on A is a function $d : A \to [0,1] \cap \mathbb{Q}$ (the rationals) such that $\sum_{a \in A} d(a) = 1$. A probability distribution is *Dirac* if it assigns 1 to one element and 0 to all the others. The *support* of a probability distribution d is the set $\mathit{support}(d) := \{a \in A : d(a) > 0\}$. The set of all probability distributions on A is denoted by $\mathcal{D}(A)$.

2.1 Probabilistic Transition Systems.

A *probabilistic labelled transition system (pLTS)* is a tuple $\mathcal{S} = (S, \Sigma, \to)$, where S is a finite or countable set of *states*, Σ is a finite input *alphabet*, and $\to \subseteq S \times \Sigma \times \mathcal{D}(S)$ is a *transition relation*. We write $s \xrightarrow{a} d$ to say that $(s, a, d) \in \to$. We also write $s \to s'$ to say that there exists $s \xrightarrow{a} d$ with $s' \in \mathit{support}(d)$. We assume that $\mathcal{S}$ is finitely branching, i.e., each state s has finitely many transitions $s \xrightarrow{a} d$. In general a pLTS combines probabilistic and non-deterministic branching. A pLTS is said to be *fully probabilistic* if for each state $s \in S$ and action $a \in \Sigma$ we have $s \xrightarrow{a} d$ for at most one distribution d. Given a fully probabilistic pLTS, we write $s \xrightarrow{a,x} s'$ to say that there is $s \xrightarrow{a} d$ such that $d(s') = x$.

Let $\mathcal{S} = (S, \Sigma, \to)$ be a pLTS and R an equivalence relation on S. We say that two distributions $d, d' \in \mathcal{D}(S)$ are *R-equivalent* if for all R-equivalence classes E, $\sum_{s \in E} d(s) = \sum_{s \in E} d'(s)$. We furthermore say that R is a *bisimulation relation* if $s \; R \; t$ implies that for each action $a \in \Sigma$ and each transition $s \xrightarrow{a} d$ there is a transition $t \xrightarrow{a} d'$ such that d and d' are R-equivalent. The union of all bisimulation relations of $\mathcal{S}$ is itself a bisimulation relation. This relation is called *bisimilarity* and is denoted $\sim$ [14].

We also have the following inductive characterisation of bisimilarity. Define a decreasing sequence of equivalence relations $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots$ by putting $s \sim_0 t$ for all s, t, and $s \sim_{n+1} t$ if and only if for all $a \in \Sigma$ and $s \xrightarrow{a} d$ there is $t \xrightarrow{a} d'$ such that $\sum_{s \in E} d(s) = \sum_{s \in E} d'(s)$ for all $\sim_n$-equivalence classes E. It is then straightforward that the sequence $\sim_n$ converges to $\sim$, i.e., $\bigcap_{n \in \mathbb{N}} \sim_n \; = \; \sim$.

2.2 Probabilistic Pushdown Automata.

A *probabilistic pushdown automaton* (pPDA) is a tuple $\Delta = (Q, \Gamma, \Sigma, \hookrightarrow)$ where Q is a finite set of *states*, Γ is a finite *stack alphabet*, Σ is a finite *input alphabet*, and $\hookrightarrow \subseteq Q \times \Gamma \times \Sigma \times \mathcal{D}(Q \times \Gamma^{\leq 2})$ (with $\Gamma^{\leq 2} := \{\varepsilon\} \cup \Gamma \cup (\Gamma \times \Gamma)$) (where ε denotes the empty string).

When speaking of the *size* of Δ, we assume that the probabilities in the transition relation are given as quotients of integers written in binary. A tuple $(q, X) \in Q \times \Gamma$ is called a *head*. A pPDA is *fully probabilistic* if for each head (q, X) and action $a \in \Sigma$ there is at most one distribution d with $(q, X, a, d) \in \hookrightarrow$. A configuration of a pPDA is an element $(q, \beta) \in Q \times \Gamma^*$, and we sometimes write just $q\beta$ instead of (q, β). We write $qX \overset{a}{\hookrightarrow} d$ to denote $(q, X, a, d) \in \hookrightarrow$, that is, in a control state q with X at the top of the stack the pPDA makes an a-transition to the distribution d. In a fully probabilistic pPDA we also write $qX \xhookrightarrow{a,x} r\beta$ if $qX \overset{a}{\hookrightarrow} d$ and $d(r\beta) = x$.

A *probabilistic basic process algebra (pBPA)* Δ is a pPDA with only one control state. In this case we sometimes omit the control state from the representation of a configuration. A *probabilistic one-counter automaton (pOCA)* is a pPDA with a stack alphabet containing only two symbols X and Z, where the transition function is restricted so that Z always and only occurs at the bottom of the stack. A *probabilistic visibly pushdown automaton (pvPDA)* is a pPDA with a partition of the actions $\Sigma = \Sigma_r \cup \Sigma_{int} \cup \Sigma_c$ such that for all $pX \overset{a}{\hookrightarrow} d$ we have: if $a \in \Sigma_r$ then $\mathit{support}(d) \subseteq Q \times \{\varepsilon\}$; if $a \in \Sigma_{int}$ then $\mathit{support}(d) \subseteq Q \times \Gamma$; if $a \in \Sigma_c$ then $\mathit{support}(d) \subseteq Q \times (\Gamma \times \Gamma)$.

A pPDA $\Delta = (Q, \Gamma, \Sigma, \hookrightarrow)$ generates a pLTS $\mathcal{S}(\Delta) = (Q \times \Gamma^*, \Sigma, \to)$ as follows. For each $\beta \in \Gamma^*$ a rule $qX \overset{a}{\hookrightarrow} d$ of Δ induces a transition $qX\beta \xrightarrow{a} d'$ in $\mathcal{S}(\Delta)$, where $d' \in \mathcal{D}(Q \times \Gamma^*)$ is defined by $d'(p\alpha\beta) = d(p\alpha)$ for all $p \in Q$ and $\alpha \in \Gamma^*$. Note that all configurations with the empty stack define terminating states of $\mathcal{S}(\Delta)$.

The *bisimilarity problem* asks whether two configurations $q_1\alpha_1$ and $q_2\alpha_2$ of a given pPDA Δ are bisimilar when regarded as states of the induced pLTS $\mathcal{S}(\Delta)$.

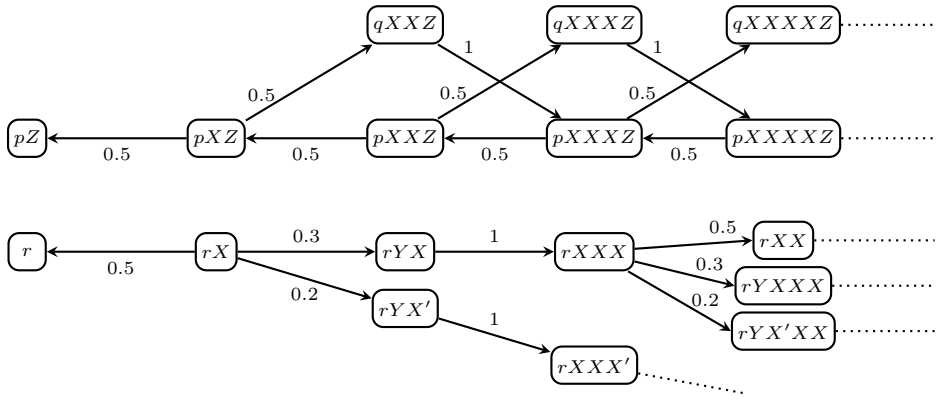


Figure 1 A fragment of $\mathcal{S}(\Delta)$ from Example 1.

▶ **Example 1.** Consider the fully probabilistic pPDA $\Delta = (\{p, q, r\}, \{X, X', Y, Z\}, \{a\}, \hookrightarrow)$ with the following rules (omitting the unique action a):

$$pX \overset{0.5}{\hookrightarrow} qXX, \qquad pX \overset{0.5}{\hookrightarrow} p, \qquad qX \overset{1}{\hookrightarrow} pXX,$$
$$rX \overset{0.3}{\hookrightarrow} rYX, \qquad rX \overset{0.2}{\hookrightarrow} rYX', \qquad rX \overset{0.5}{\hookrightarrow} r, \qquad rY \overset{1}{\hookrightarrow} rXX,$$
$$rX' \overset{0.4}{\hookrightarrow} rYX, \qquad rX' \overset{0.1}{\hookrightarrow} rYX', \qquad rX' \overset{0.5}{\hookrightarrow} r.$$

The restriction of Δ to the control states p, q and to the stack symbols X, Z yields a pOCA. The restriction of Δ to the control state r and the stack symbols X, X', Y yields a pBPA. A fragment of the pLTS $\mathcal{S}(\Delta)$ is shown in Figure 1. The configurations pXZ and rX are bisimilar, as there is a bisimulation relation with equivalence classes $\{pX^kZ\} \cup \{rw \mid w \in \{X, X'\}^k\}$ for all $k \geq 0$ and $\{qX^{k+1}Z\} \cup \{rYw \mid w \in \{X, X'\}^k\}$ for all $k \geq 1$.

3 From Probabilistic to Nondeterministic Bisimilarity

A nondeterministic pushdown automaton (PDA) is a special case of a probabilistic pushdown automaton in which the transition function assigns only Dirac distributions. We give a novel reduction of the bisimilarity problem for pPDA to the bisimilarity problem for PDA. Because the latter is known to be decidable [15], we get decidability of the bisimilarity problem for pPDA.

As a first step we give the following characterisation of R-equivalence of two distributions (defined earlier).

▶ **Lemma 2.** *Let R be an equivalence relation on a set S. Two distributions d, d' on S are R-equivalent if and only if for all $A \subseteq S$ we have $d(A) \leq d'(R(A))$, where $R(A)$ denotes the image of A under R.*

Proof. For the *if* direction we reason as follows. For each equivalence class E we have $d(E) \leq d'(E)$. But since d and d' have total mass 1 we must have $d(E) = d'(E)$ for all equivalence classes E.

Conversely if d and d' are R-equivalent. Then $d(A) \leq d(R(A)) = d'(R(A))$ for any set A, since $R(A)$ is a countable union of equivalence classes. ◀

We now give our reduction. Let $\Delta = (Q, \Gamma, \Sigma, \hookrightarrow)$ be a pPDA and $q_1\gamma_1$, $q_2\gamma_2$ two configurations of Δ. We define a new PDA $\Delta' = (Q, \Gamma', \Sigma', \circ\!\!\rightarrow)$ that extends Δ with extra

stack symbols, input letters and transition rules. In particular, a configuration of Δ can also be regarded as a configuration of Δ'. The definition of Δ' is such that two Δ-configurations $q_1\gamma_1$ and $q_2\gamma_2$ are bisimilar in Δ if and only if the same two configurations are bisimilar in Δ'.

Intuitively we eliminate probabilistic transitions by treating probabilities as part of the input alphabet. To this end, let $W \subseteq \mathbb{Q}$ be the set of rational numbers of the form $d(A)$ for some rule $pX \overset{a}{\hookrightarrow} d$ in Δ and $A \subseteq \mathit{support}(d)$. Think of W as the set of relevant transition weights.

We define Δ' as follows. Note that when defining rules of Δ' we write just $q\gamma$ instead of the Dirac distribution assigning 1 to $q\gamma$.

- The stack alphabet Γ' contains all symbols from Γ. In addition, for every rule $pX \overset{a}{\hookrightarrow} d$ in Δ it contains a new symbol $\langle d\rangle$ and for every $T \subseteq \mathit{support}(d)$ a symbol $\langle T\rangle$.
- The input alphabet Σ' is equal to $\Sigma \cup W \cup \{\#\}$ where $\#$ is a distinguished action not in Σ or W.
- The transition function $\circ\!\!\rightarrow$ is defined as follows. For every rule $qX \overset{a}{\hookrightarrow} d$, there is a rule $qX \overset{a}{\circ\!\!\rightarrow} q\langle d\rangle$. We also have a rule $q\langle d\rangle \overset{w}{\circ\!\!\rightarrow} q\langle T\rangle$ if $T \subseteq \mathit{support}(d)$ and $d(T) \geq w \in W$. Finally, we have a rule $q\langle T\rangle \overset{\#}{\circ\!\!\rightarrow} p\alpha$ if $p\alpha \in T$.

The PDA Δ' can be constructed in time exponential in the size of Δ, and in polynomial time if the branching degree of Δ is bounded (i.e. if we fix a number N and consider only pPDAs with branching degree at most N). The analysis can be found in [7]. The correctness of the construction is captured by the following lemma and proved in [7].

▶ **Lemma 3.** *For any configurations $q_1\gamma_1, q_2\gamma_2$ of Δ we have $q_1\gamma_1 \sim q_2\gamma_2$ in Δ if and only if $q_1\gamma_1 \sim q_2\gamma_2$ in Δ'.*

Let us show intuitively why bisimilar configurations in Δ remain bisimilar considered as configurations of Δ'. Every computation step of Δ is simulated in three steps by Δ'. Let $q_1X_1\gamma_1$ and $q_2X_2\gamma_2$ be bisimilar configurations of Δ. Then in Δ' a transition of $q_1X_1\gamma_1$ to $q_1\langle d_1\rangle\gamma_1$ can be matched by a transition (under the same action) of $q_2X_2\gamma_2$ to $q_2\langle d_2\rangle\gamma_2$ such that the distributions d_1 and d_2 are $\sim$-equivalent (and *vice versa*). In particular, by Lemma 2, for any set of configurations T the set T' obtained by saturating T under bisimilarity is such that $d_1(T) \leq d_2(T')$. Let $\bar{T}$ and $\bar{T}'$ respectively contain the elements of T and T' from which the suffixes γ_1 and γ_2 are removed. Then, as a second step of simulation of Δ by Δ', a transition of $q_1\langle d_1\rangle\gamma_1$ to a state $q_1\langle\bar{T}\rangle\gamma_1$ with label $w \in W$ can be matched by a transition of Δ' to $q_2\langle\bar{T}'\rangle\gamma_2$ with the same label (similarly any transition of $q_2\langle d_2\rangle\gamma_2$ can be matched by a transition of $q_1\langle d_1\rangle\gamma_1$). Finally, as T and T' contain elements from the same bisimilarity equivalence classes, in the third step a $\#$-transition from $q_1\langle\bar{T}\rangle\gamma_1$ to some $q_1'\alpha_1\gamma_1$ can be matched by a $\#$-transition of $q_2\langle\bar{T}'\rangle\gamma_2$ to $q_2'\alpha_2\gamma_2$ such that $q_1'\alpha_1\gamma_1$ and $q_2'\alpha_2\gamma_2$ are again bisimilar in Δ (and *vice versa*).

The three steps are illustrated in Figure 2, where the successors of the configurations pXZ and rX in the system $\mathcal{S}(\Delta')$ for the PDA Δ' constructed from the pPDA Δ from Example 1 are drawn.

Lemma 3 gives rise to the following theorem.

▶ **Theorem 4.** *For any pPDA Δ there is a PDA Δ' constructible in exponential time such that for any configurations $q_1\gamma_1, q_2\gamma_2$ of Δ we have $q_1\gamma_1 \sim q_2\gamma_2$ in Δ if and only if $q_1\gamma_1 \sim q_2\gamma_2$ in Δ'. In addition, if Δ is a pBPA, then Δ' is a BPA.*

Using Theorem 4 and [15, 4], we get the following corollary.

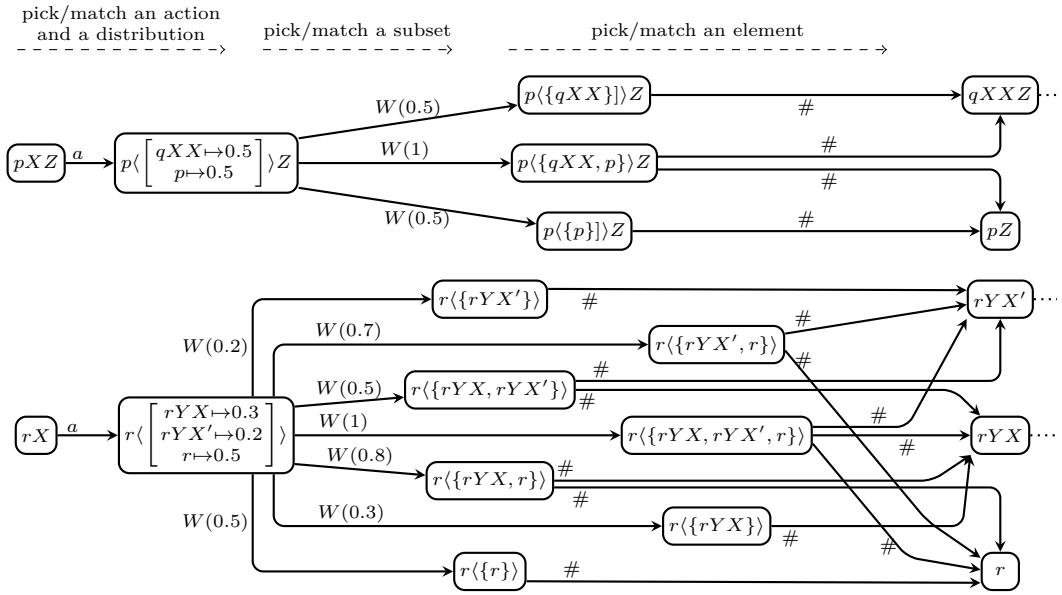


Figure 2 An example of the construction for Lemma 3. Here, an arrow labelled $W(x)$ is an abbreviation for multiple transitions labelled by all multiples of 0.1 between 0.1 and x.

▶ **Corollary 5.** *The bisimilarity problem for pPDA is decidable, and the bisimilarity problem for pBPA is decidable in triply exponential time.*

4 Upper Bounds

4.1 Bisimilarity of pOCA is in PSPACE

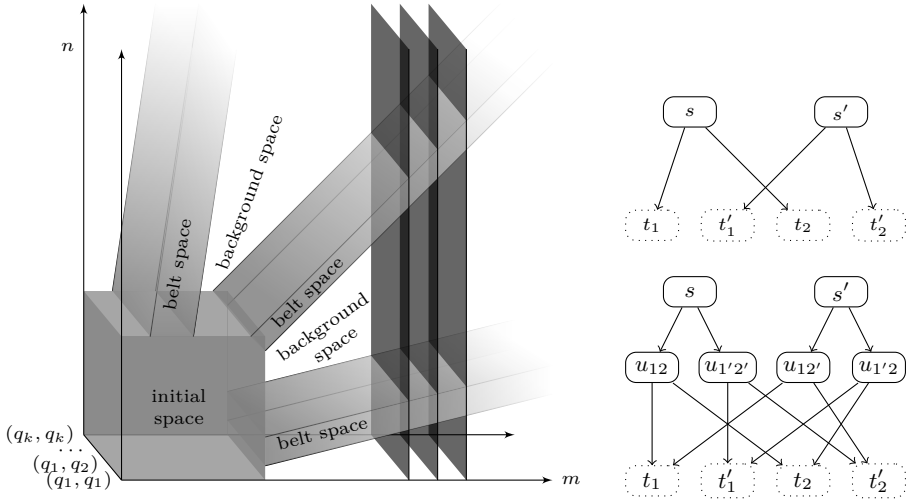
The bisimilarity problem for (non-probabilistic) one-counter automata is PSPACE-complete, as shown in [2]. It turns out that for pOCA we get PSPACE-completeness as well. The lower bound is shown in Section 5; here we show:

▶ **Theorem 6.** *The bisimilarity problem for pOCA is in PSPACE, even if we present the instance $\Delta = (Q, \{Z, X\}, \Sigma, \hookrightarrow)$, pX^mZ, qX^nZ (for which we ask if $pX^mZ \sim qX^nZ$) by a shorthand using m, n written in binary.*

The reduction underlying Theorem 4 would only provide an exponential-space upper bound, so we give a pOCA-specific polynomial-space algorithm. In fact, we adapt the algorithm from [2]; the principles are the same but some ingredients have to be slightly modified. The following text is meant to give the idea in a self-contained manner, though at a more abstract level than in [2]. The main difference is in the notion of local consistency, discussed around Proposition 11.

Similarly as [2], we use a geometrical presentation of relations on the set of configurations (Fig. 3(a) reflects such a presentation). A relation can be identified with a 1/0 (or YES/NO) colouring of the "grid" $\mathbb{N} \times \mathbb{N} \times (Q \times Q)$:

▶ **Definition 7.** For a relation R on $Q \times (\{X\}^*Z)$, by the *(characteristic) colouring* χ_R we mean the function $\chi_R : \mathbb{N} \times \mathbb{N} \times (Q \times Q) \to \{1, 0\}$ where $\chi_R(m, n, (p, q)) = 1$ if and only if $(pX^mZ, qX^nZ) \in R$. Given (a colouring) $\chi : \mathbb{N} \times \mathbb{N} \times (Q \times Q) \to \{1, 0\}$, by R_χ we denote the relation $R_\chi = \{(pX^mZ, qX^nZ) \mid \chi(m, n, (p, q)) = 1\}$.



(a) Partition of a grid, and a moving vertical window of width 3

(b) AND-gadget (top) and OR-gadget (bottom)

Figure 3 Figures for Section 4.1 (left) and 5 (right)

The algorithm uses the fact that $\chi_\sim$ is "regular", i.e. $\{(m,n,(p,q)) \mid pX^mZ \sim qX^nZ\}$ is a (special) semilinear set. More concretely, there are polynomials $pol_1, pol_2 : \mathbb{N} \to \mathbb{N}$ (independent of the pOCA Δ) such that the following partition of the grid $\mathbb{N} \times \mathbb{N} \times (Q \times Q)$ (sketched in Fig. 3(a)) has an important property specified later. If $Q = \{q_1, q_2, \ldots, q_k\}$, hence $|Q| = k$, then the grid is partitioned into three parts: the *initial-space*, i.e. $\{(m,n,(p,q)) \mid m,n \leq pol_2(k)\}$, the *belt-space*, which is given by at most k^4 linear belts, with the slopes $\frac{c}{d}$ where $c,d \in \{1,2,\ldots,k^2\}$ and with the (vertical) thickness bounded by $pol_1(k)$, and the rest, called the *background*. Moreover, $pol_2(k)$ is sufficiently large w.r.t. $pol_1(k)$, so that the belts are separated by the background outside the initial space.

The mentioned important property is that there is a period ψ, given by an exponential function of k, such that if two points $(m,n,(p,q))$ and $(m+i\psi, n+j\psi, (p,q))$ (for $i,j \in \mathbb{N}$) are both in the background, for both m,n larger then a polynomial bound, then $\chi_\sim$ has the same value for both these points; in other words, $\chi_\sim$ colours the background periodically. Another important ingredient is the locality of the bisimulation conditions, resulting from the fact that the counter value can change by at most 1 per step.

To explain the "grid-partition", we start with considering the finite automaton $\mathcal{F}_\Delta$ underlying Δ; $\mathcal{F}_\Delta$ behaves like Δ "pretending" that the counter is always positive.

▶ **Definition 8.** For a pOCA $\Delta = (Q, \{Z,X\}, \Sigma, \hookrightarrow)$, in the *underlying finite pLTS* $\mathcal{F}_\Delta = (Q, \Sigma, \to)$ we have a transition $p \xrightarrow{a} d'$ if and only if there is a transition $pX \overset{a}{\hookrightarrow} d$ such that $d'(q) = d(q,\varepsilon) + d(q,X) + d(q,XX)$ (for all $q \in Q$).

Using standard partition-refinement arguments, we observe that $\sim_{k-1} = \sim_k = \sim$ on $\mathcal{F}_\Delta$ when $k = |Q|$. For configurations of Δ we now define the distance dist to the set of configurations which are "INCompatible" with $\mathcal{F}_\Delta$.

▶ **Definition 9.** Assuming a pOCA $\Delta = (Q, \{Z,X\}, \Sigma, \hookrightarrow)$, where $|Q| = k$, we define $\mathsf{INC} \subseteq Q \times (\{X\}^*Z)$ and $\mathsf{dist} : Q \times (\{X\}^*Z) \to \mathbb{N} \cup \{\infty\}$ as follows:

- $\mathsf{INC} = \{pX^mZ \mid \forall q \in Q : pX^mZ \not\sim_k q\}$ (where q is a state in $\mathcal{F}_\Delta$),
- $\mathsf{dist}(pX^mZ) = \min\{\ell \mid \exists q\gamma \in \mathsf{INC} : pX^mZ(\hookrightarrow)^\ell q\gamma\}$; we set $\min \emptyset = \infty$.

Since $pX^mZ \sim_m p$ (by induction on m), and thus $pX^mZ \in \mathsf{INC}$ implies $m < k$, we can surely construct INC for a given pOCA in polynomial space.

▶ **Proposition 10.**

1. *If* $pX^mZ \sim qX^nZ$ *then* $\mathsf{dist}(pX^mZ) = \mathsf{dist}(qX^nZ)$.
2. *If* $\mathsf{dist}(pX^mZ) = \mathsf{dist}(qX^nZ) = \infty$ *then* $pX^mZ \sim qX^nZ$ *iff* $pX^mZ \sim_k qX^nZ$.

The proof is the same as in the non-probabilistic case. (Point 1 is obvious. For Point 2 we verify that the set $\{(q_1X^{n_1}Z, q_2X^{n_2}Z) \mid q_1X^{n_1}Z \sim_k q_2X^{n_2}Z \text{ and } \mathsf{dist}(q_1X^{n_1}Z) = \mathsf{dist}(q_2X^{n_2}Z) = \infty$ is a bisimulation.)

Consider a shortest path from pX^mZ to INC (for large m). It is not hard to prove (as in [2, Lemma 10]) that such a path can be based on iterating a simple counter-decreasing cycle (of length $\leq k$), possibly preceded by a polynomial prefix and followed by a polynomial suffix. So (finite) $\mathsf{dist}(pX^mZ)$ can be always expressed by the use of linear functions $\frac{\ell}{e}m + b$ where $\ell, e \leq k$ are the length and the decreasing effect of a simple cycle and b is bounded by a polynomial in k. It follows that if we have $\mathsf{dist}(pX^mZ) = \mathsf{dist}(qX^nZ) < \infty$, then $n = \frac{\ell_1 e_2}{e_1 \ell_2}m + b'$, which shows that $(m, n, (p, q))$ lies in one of the above mentioned belts, or in the initial space when m, n are small.

As a consequence, in the background points $(m, n, (p, q))$ we have either $\mathsf{dist}(pX^mZ) = \mathsf{dist}(qX^nZ) = \infty$, and $\chi_\sim(m, n, (p, q)) = 1$ if and only if $pX^mZ \sim_k qX^nZ$, or $\mathsf{dist}(pX^mZ) \neq \mathsf{dist}(qX^nZ)$ (and thus $\chi_\sim(m, n, (p, q)) = 0$). So we can easily compute $\chi_\sim$ for any background point in polynomial space.

The above mentioned shortest paths to INC also show that if we choose $\psi = k!$ (so $\psi = O(2^{k \log k})$) then we have $pX^mZ \to^* \mathsf{INC}$ if and only if $pX^{(m+\psi)}Z \to^* \mathsf{INC}$ (for m larger than some polynomial bound), since the counter-effect of each simple cycle divides ψ. Hence ψ is a background period as mentioned above.

A nondeterministic algorithm, verifying that $p_0X^{m_0}Z \sim q_0X^{n_0}Z$ for $(m_0, n_0, (p_0, q_0))$ in the initial or belt-space, is based on "moving a vertical window of width 3" (as depicted in Fig. 3(a)); in each phase, the window is moved by 1 (to the right), its intersection with the initial and belt space (containing polynomially many points) is computed, a colouring on this intersection is guessed ($\chi_\sim$ is intended) and its (local) consistency is checked (for which also $\chi_\sim$ on the neighbouring background points is computed). More precisely, in the first, i.e. leftmost, window position a colouring in all three (vertical) slices is guessed and the local consistency in the first two slices is checked; after any later shift of the window by one to the right, a colouring in the new (the rightmost) slice is guessed (the guesses in the previous two slices being remembered), and the consistency in the current middle slice is checked. If this is successfully performed for exponentially many steps, after $(m_0, n_0, (p_0, q_0))$ has been coloured with 1, then it is guaranteed that the algorithm could successfully run forever; the pigeonhole principle induces that each belt could be periodically coloured, with an exponential period compatible with the period of the background-border of the belt. Such a successful run of the algorithm, exponential in time but obviously only polynomial in the required space, is thus a witness of $p_0X^{m_0}Z \sim q_0X^{n_0}Z$. Since PSPACE=NPSPACE, we have thus sketched a proof of Theorem 6.

It remains to define precisely the consistency of a colouring, guaranteeing that a successful run of the algorithm really witnesses $p_0X^{m_0}Z \sim q_0X^{n_0}Z$. (As already mentioned, this is the main change wrt [2].) We use the following particular variant of characterizing (probabilistic) bisimilarity. Given a pLTS $(S, \Sigma, \to)$, we say that (s, t) is *consistent w.r.t.* a relation R on S (not necessarily an equivalence) if for each $s \xrightarrow{a} d$ there is $t \xrightarrow{a} d'$, and conversely for each $t \xrightarrow{a} d'$ there is $s \xrightarrow{a} d$, such that d, d' are R'-equivalent where R' is the least equivalence containing the set $\{(s', t') \mid s \to s', t \to t', (s', t') \in R\}$. A *relation* R is *consistent* if each

$(s,t) \in R$ is consistent w.r.t. R. The following proposition can be verified along the standard lines.

▶ **Proposition 11.** *$\sim$ is consistent. If R is consistent then $R \subseteq \sim$.*

Our algorithm can surely (locally) check the above defined consistency of the constructed χ (i.e. of R_χ).

4.2 Bisimilarity of pvPDA is in EXPTIME

It is shown in [16, Theorem 3.3] that the bisimilarity problem for (non-probabilistic) vPDA is EXPTIME-complete. We will show that the same holds for pvPDA. First we show the upper bound:

▶ **Theorem 12.** *The bisimilarity problem for pvPDA is in EXPTIME.*

In [16] the upper bound is proved using a reduction to the model-checking problem for (non-visibly) PDA and the modal μ-calculus. The latter problem is in EXPTIME by [17]. This reduction does not apply in the probabilistic case. The reduction from Section 3 cannot be directly applied either, since it incurs an exponential blowup, yielding only a double-exponential algorithm if combined with the result of [17]. Therefore we proceed as follows: First we give a direct proof for (non-probabilistic) vPDA, i.e., we show via a new proof that the bisimilarity problem for vPDA is in EXPTIME. Then we show that the reduction from Section 3 yields a non-probabilistic vPDA that is exponential only in a way that the new algorithm can be made run in single-exponential time: The crucial observation is that the reduction replaces each step in the pvPDA by three steps in the (non-probabilistic) vPDA. An exponential blowup occurs only in intermediate states of the new LTS. Our algorithm allows to deal with those states in a special pre-processing phase. See [7] for details.

5 Lower Bounds

In this section we show hardness results for pOCA and pvPDA. We start by defining two gadgets, adapted from [5], that will be used for both results. The gadgets are pLTS that allow us to simulate AND and OR gates using probabilistic bisimilarity. We depict the gadgets in Figure 3(b), where we assume that all edges have probability 1/2 and have the same label. The gadgets satisfy the following propositions (here $s \xrightarrow{a} t_1 \mid t_2$ is a shorthand for $s \xrightarrow{a} d$ where $d(t_1) = d(t_2) = 0.5$).

▶ **Proposition 13.** (AND-gadget) *Suppose s, s', t_1, t_1', t_2, t_2' are states in a pLTS such that $t_1 \not\sim t_2'$ and the only transitions outgoing from s, s' are $s \xrightarrow{a} t_1 \mid t_2$ and $s' \xrightarrow{a} t_1' \mid t_2'$. Then $s \sim s'$ if and only if $t_1 \sim t_1' \wedge t_2 \sim t_2'$.*

▶ **Proposition 14.** (OR-gadget) *Suppose s, s', t_1, t_1', t_2, t_2', and u_{12}, $u_{1'2}$, $u_{12'}$, $u_{1'2'}$ are states in a pLTS. Let the only transitions outgoing from $s, s', u_{12}, u_{1'2}, u_{12'}, u_{1'2'}$ be*

$$s \xrightarrow{a} u_{12} \mid u_{1'2'}\ ,\ s' \xrightarrow{a} u_{12'} \mid u_{1'2}\ ,$$
$$u_{12} \xrightarrow{a} t_1 \mid t_2\ ,\ u_{1'2'} \xrightarrow{a} t_1' \mid t_2'\ ,\ u_{12'} \xrightarrow{a} t_1 \mid t_2'\ ,\ u_{1'2} \xrightarrow{a} t_1' \mid t_2\ .$$

Then $s \sim s'$ if and only if $t_1 \sim t_1' \vee t_2 \sim t_2'$.

5.1 Bisimilarity of pOCA is PSPACE-hard

In this section we prove the following:

▶ **Theorem 15.** *Bisimilarity for pOCA is PSPACE-hard, even for unary (i.e., with only one action) and fully probabilistic pOCA, and for fixed initial configurations of the form* pXZ, qXZ.

In combination with Theorem 6 we obtain:

▶ **Corollary 16.** *The bisimilarity problem for pOCA is PSPACE-complete.*

Proof of Theorem 15. We use a reduction from the emptiness problem for alternating finite automata with a one-letter alphabet, known to be PSPACE-complete [9, 10]; our reduction resembles the reduction in [16] for (non-probabilistic) visibly one-counter automata.

A *one-letter alphabet alternating finite automaton*, 1L-AFA, is a tuple $A = (Q, \delta, q_0, F)$ where Q is the (finite) set of *states*, q_0 is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and the *transition function* δ assigns to each $q \in Q$ either $q_1 \wedge q_2$ or $q_1 \vee q_2$, where $q_1, q_2 \in Q$.

We define the predicate $Acc \subseteq Q \times \mathbb{N}$ by induction on the second component (i.e. the length of a one-letter word); $Acc(q, n)$ means "A starting in q accepts n": $Acc(q, 0)$ if and only if $q \in F$; $Acc(q, n{+}1)$ if and only if either $\delta(q) = q_1 \wedge q_2$ and we have both $Acc(q_1, n)$ and $Acc(q_2, n)$, or $\delta(q) = q_1 \vee q_2$ and we have $Acc(q_1, n)$ or $Acc(q_2, n)$.

The *emptiness problem for 1L-AFA* asks, given a 1L-AFA A, if the set $\{n \mid Acc(q_0, n)\}$ is empty.

We reduce the emptiness of 1L-AFA to our problem. We thus assume a 1L-AFA (Q, δ, q_0, F), and we construct a pOCA Δ as follows. Δ has $2|Q| + 3$ 'basic' states; the set of basic states is $\{p, p', r\} \cup Q \cup Q'$ where $Q' = \{q' \mid q \in Q\}$ is a copy of Q and r is a special dead state. Additional auxiliary states will be added to implement AND- and OR-gadgets. Δ will have only one input letter, denoted a, and will be fully probabilistic.

We aim to achieve $pXZ \sim p'XZ$ if and only if $\{n \mid Acc(q_0, n)\}$ is empty; another property will be that

$$qX^nZ \sim q'X^nZ \text{ if and only if } \neg Acc(q, n). \tag{1}$$

For each $q \in F$ we add a transition $qZ \xrightarrow{a} d$ where $d(r, Z) = 1$, but qZ is dead (i.e., there is no transition $qZ \xrightarrow{a} ..$) if $q \notin F$; $q'Z$ is dead for any $q' \in Q'$. Both rX and rZ are dead as well. Hence (1) is satisfied for $n = 0$. Now we show (1) holds for $n > 0$.

For q with $\delta(q) = q_1 \vee q_2$ we implement an AND-gadget from Figure 3(b) (top) guaranteeing $qX^{n+1}Z \sim q'X^{n+1}Z$ if and only if $q_1X^nZ \sim q_1'X^nZ$ and $q_2X^nZ \sim q_2'X^nZ$ (since $\neg Acc(q, n{+}1)$ if and only if $\neg Acc(q_1, n)$ and $\neg Acc(q_2, n)$):

We add rules $qX \to r_1X \mid r_2X$ (this is a shorthand for $qX \xrightarrow{a} [r_1X \mapsto 0.5, r_2X \mapsto 0.5]$) and $q'X \to r_1'X \mid r_2'X$,
and also $r_1X \to q_1 \mid s_1X$, $r_2X \to q_2 \mid s_2X$, $r_1'X \to q_1' \mid s_1X$, $r_2'X \to q_2' \mid s_2X$,
and $s_1X \xrightarrow{0.5} s_1X$, $s_1X \xrightarrow{0.5} r$, $s_2X \xrightarrow{0.4} s_2X$, $s_2X \xrightarrow{0.6} r$. The intermediate states r_1, r_2, r_1', r_2', and s_1, s_2 serve to implement the condition $t_1 \not\sim t_2'$ from Proposition 13.

For q with $\delta(q) = q_1 \wedge q_2$ we (easily) implement an OR-gadget from Figure 3(b) (bottom) guaranteeing $qX^{n+1}Z \sim q'X^{n+1}Z$ if and only if $q_1X^nZ \sim q_1'X^nZ$ or $q_2X^nZ \sim q_2'X^nZ$ (since $\neg Acc(q, n{+}1)$ if and only if $\neg Acc(q_1, n)$ or $\neg Acc(q_2, n)$).

To finish the construction, we add transitions $pX \xrightarrow{a} d$ where $d(p, X^2) = d(q_0, \varepsilon) = d(r, X) = \frac{1}{3}$ and $p'X \xrightarrow{a} d'$ where $d'(p', X^2) = d(q_0', \varepsilon) = d(r, X) = \frac{1}{3}$; the transitions added before guarantee that $pX^{n+2}Z \not\sim q_0'X^nZ$ and $q_0X^nZ \not\sim p'X^{n+2}Z$. ◀

5.2 Bisimilarity of pvPDA is EXPTIME-hard

In this section we prove the following:

▶ **Theorem 17.** *Bisimilarity for pvPDA is EXPTIME-hard, even for fully probabilistic pvPDA with* $|\Sigma_r| = |\Sigma_{int}| = |\Sigma_c| = 1$.

In combination with Theorem 12 we obtain:

▶ **Corollary 18.** *The bisimilarity problem for pvPDA is EXPTIME-complete.*

It was shown in [16] that bisimilarity for (non-probabilistic) vPDA is EXPTIME-complete. The hardness result there follows by observing that the proof given in [13] for general PDA works in fact even for vPDA. Referring to the conference version of [13], it is commented in [16]: "Though conceptually elegant, the technical details of the reduction are rather tedious." For those reasons we give a full reduction from the problem of determining the winner in a reachability game on pushdown processes. This problem was shown EXPTIME-complete in [17]. Our reduction proves Theorem 17, i.e., for unary and fully probabilistic pvPDA, and at the same time provides a concise proof for (non-probabilistic) vPDA.

Proof of Theorem 17. Let $\Delta = (Q, \Gamma, \{a\}, \hookrightarrow)$ be a unary non-probabilistic PDA with a control state partition $Q = Q_0 \cup Q_1$ and an initial configuration p_0X_0. We call a configuration $pX\alpha$ *dead* if it has no successor configuration, i.e., if Δ does not have a rule with pX on the left-hand side. Consider the following game between Player 0 and Player 1 on the LTS $\mathcal{S}(\Delta)$ induced by Δ: The game starts in p_0X_0. Whenever the game is in a configuration $p\alpha$ with $p \in Q_i$ (where $i \in \{0,1\}$), Player i chooses a successor configuration of $p\alpha$ in $\mathcal{S}(\Delta)$. The goal of Player 1 is to reach a dead configuration; the goal of Player 0 is to avoid that. It is shown in [17, pp. 261–262] that determining the winner in that game is EXPTIME-hard.

W.l.o.g. we can assume that each configuration has at most two successor configurations, and that no configuration with empty stack is reachable. We construct a fully probabilistic pvPDA $\bar{\Delta} = (\bar{Q}, \Gamma, \{a_r, a_{int}, a_c\}, \circ\!\!\rightarrow)$ such that the configurations p_0X_0 and p'_0X_0 of $\bar{\Delta}$ are bisimilar if and only if Player 0 can win the game. For each control state $p \in Q$ the set $\bar{Q}$ includes p and a copy p'.

For each $pX \in Q \times \Gamma$, if pX is dead in Δ, we add a rule $pX \overset{a_{int},1}{\circ\!\!\longrightarrow} pX$ in $\bar{\Delta}$, and a rule $p'X \overset{a_{int},1}{\circ\!\!\longrightarrow} zX$ where $z \in \bar{Q}$ is a special control state not occurring on any left-hand side. This ensures that if pX is dead in Δ (and hence Player 1 wins), then we have $pX \not\sim p'X$ in $\bar{\Delta}$.

For each $pX \in Q \times \Gamma$ that has in Δ a single successor configuration $q\alpha$, we add rules $pX \overset{a,1}{\circ\!\!\longrightarrow} q\alpha$ and $p'X \overset{a,1}{\circ\!\!\longrightarrow} q'\alpha$, where $a = a_r, a_{int}, a_c$ if $|\alpha| = 0, 1, 2$, respectively.

For each $pX \in Q \times \Gamma$ that has in Δ two successor configurations, let $p_1\alpha_1$ and $p_2\alpha_2$ denote the successor configurations. W.l.o.g. we can assume that $\alpha_1 = X_1 \in \Gamma$ and $\alpha_2 = X_2 \in \Gamma$.

- If $p \in Q_0$ we implement an OR-gadget from Figure 3(b): let $(p_1X_1p_2X_2), (p'_1X_1p'_2X_2), (p_1X_1p'_2X_2), (p'_1X_1p_2X_2) \in \bar{Q}$ be fresh control states, and add rules $pX \circ\!\!\rightarrow (p_1X_1p_2X_2)X \mid (p'_1X_1p'_2X_2)X$ (this is a shorthand for $pX \overset{a_{int},0.5}{\circ\!\!\longrightarrow} (p_1X_1p_2X_2)X$ and $pX \overset{a_{int},0.5}{\circ\!\!\longrightarrow} (p'_1X_1p'_2X_2)X$) and $p'X \circ\!\!\rightarrow (p_1X_1p'_2X_2)X \mid (p'_1X_1p_2X_2)X$ as well as $(p_1X_1p_2X_2)X \circ\!\!\rightarrow p_1X_1 \mid p_2X_2$ and $(p'_1X_1p'_2X_2)X \circ\!\!\rightarrow p'_1X_1 \mid p'_2X_2$ and $(p_1X_1p'_2X_2)X \circ\!\!\rightarrow p_1X_1 \mid p'_2X_2$ and $(p'_1X_1p_2X_2)X \circ\!\!\rightarrow p'_1X_1 \mid p_2X_2$.
- If $p \in Q_0$ we implement an AND-gadget from Figure 3(b): let $(p_1X_1), (p'_1X_1), (p_2X_2), (p'_2X_2) \in \bar{Q}$ be fresh control states, and add rules

$pX \circ\!\!\to (p_1X_1)X \mid (p_2X_2)X$ and $p'X \circ\!\!\to (p'_1X_1)X \mid (p'_2X_2)X$ as well as $(p_1X_1)X \overset{a_{int},1}{\circ\!\!\longrightarrow} p_1X_1$ and $(p'_1X_1)X \overset{a_{int},1}{\circ\!\!\longrightarrow} p'_1X_1$ and $(p_2X_2)X \circ\!\!\to p_2X_2 \mid zX$ and $(p'_2X_2)X \circ\!\!\to p'_2X_2 \mid zX$. Here, the transitions to zX serve to implement the condition $t_1 \not\sim t'_2$ from Proposition 13.

An induction argument now easily establishes that $p_0X_0 \sim p'_0X_0$ holds in $\bar{\Delta}$ if and only if Player 0 can win the game in Δ.

We remark that exactly the same reduction works for non-probabilistic vPDA, if the probabilistic branching is replaced by nondeterministic branching. ◀

Acknowledgements. The authors thank anonymous referees for their helpful feedback. Vojtěch Forejt is supported by a Newton International Fellowship of the Royal Society. Petr Jančar is supported by the Grant Agency of the Czech Rep. (project GAČR:P202/11/0340); his short visit at Oxford was also supported by ESF-GAMES grant no. 4513. Stefan Kiefer is supported by the EPSRC.

References

1 C. Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In *CAV*, pages 50–61, 1996.

2 S. Böhm, S. Göller, and P. Jančar. Bisimilarity of one-counter processes is PSPACE-complete. In *CONCUR*, volume 6269 of *LNCS*, pages 177–191, 2010.

3 T. Brázdil, A. Kučera, and O. Stražovský. Deciding probabilistic bisimilarity over infinite-state probabilistic systems. *Acta Inf.*, 45(2):131–154, 2008.

4 O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 545–623. North-Holland, 2001.

5 D. Chen, F. van Breugel, and J. Worrell. On the complexity of computing probabilistic bisimilarity. In *FoSSaCS*, volume 7213 of *LNCS*, pages 437–451, 2012.

6 K. Etessami, D. Wojtczak, and M. Yannakakis. Quasi-birth-death processes, tree-like QBDs, probabilistic 1-counter automata, and pushdown systems. *Perform. Eval.*, 67(9):837–857, 2010.

7 V. Forejt, P. Jančar, S. Kiefer, and J. Worrell. Bisimilarity of probabilistic pushdown automata. Technical report, arxiv.org, 2012. Available at `http://arxiv.org/abs/1210.2273`.

8 H. Fu and J.-P. Katoen. Deciding probabilistic simulation between probabilistic pushdown automata and finite-state systems. In *FSTTCS*, pages 445–456, 2011.

9 M. Holzer. On emptiness and counting for alternating finite automata. In *Developments in Language Theory*, pages 88–97, 1995.

10 P. Jančar and Z. Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *Inf. Process. Lett.*, 104(5):164–167, 2007.

11 P. Jančar. Bisimilarity on Basic Process Algebra is in 2-ExpTime (an explicit proof). *CoRR*, abs/1207.2479, 2012.

12 S. Kiefer. BPA bisimilarity is EXPTIME-hard. *CoRR*, abs/1205.7041, 2012.

13 A. Kučera and R. Mayr. On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Information and Computation*, 208(7):772–796, 2010.

14 R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR*, volume 836 of *LNCS*, pages 481–496. Springer, 1994.

15 G. Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM J. Comput.*, 34(5):1025–1106, 2005.

16 J. Srba. Beyond language equivalence on visibly pushdown automata. *Logical Methods in Computer Science*, 5(1):2, 2009.

17 I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.

Average Case Analysis of the Classical Algorithm for Markov Decision Processes with Büchi Objectives*

Krishnendu Chatterjee[1], Manas Joglekar[2], and Nisarg Shah[3]

1 **IST Austria (Institute of Science and Technology Austria)**
2 **Stanford University**
3 **Carnegie Mellon University**

Abstract

We consider Markov decision processes (MDPs) with specifications given as Büchi (liveness) objectives. We consider the problem of computing the set of *almost-sure* winning vertices from where the objective can be ensured with probability 1. We study for the first time the average case complexity of the classical algorithm for computing the set of almost-sure winning vertices for MDPs with Büchi objectives. Our contributions are as follows: First, we show that for MDPs with constant out-degree the expected number of iterations is at most logarithmic and the average case running time is linear (as compared to the worst case linear number of iterations and quadratic time complexity). Second, for the average case analysis over all MDPs we show that the expected number of iterations is constant and the average case running time is linear (again as compared to the worst case linear number of iterations and quadratic time complexity). Finally we also show that given that all MDPs are equally likely, the probability that the classical algorithm requires more than constant number of iterations is exponentially small.

1998 ACM Subject Classification D.2.4 Formal methods

Keywords and phrases MDPs, Büchi objectives, Average case analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.461

1 Introduction

Markov decision processes. *Markov decision processes (MDPs)* are standard models for probabilistic systems that exhibit both probabilistic and nondeterministic behavior [13], and widely used in verification of probabilistic systems [1, 15]. MDPs have been used to model and solve control problems for stochastic systems [12]: there, nondeterminism represents the freedom of the controller to choose a control action, while the probabilistic component of the behavior describes the system response to control actions. MDPs have also been adopted as models for concurrent probabilistic systems [7], probabilistic systems operating in open environments [18], under-specified probabilistic systems [2], and applied in diverse domains [15]. A *specification* describes the set of desired behaviors of the system, which in the verification and control of stochastic systems is typically an ω-regular set of paths. The class of ω-regular languages extends classical regular languages to infinite strings, and provides a robust specification language to express all commonly used specifications, such

* The research was supported by FWF Grant No P 23499-N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 461–473

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

as safety, liveness, fairness, etc [20]. Parity objectives are a canonical way to define such ω-regular specifications. Thus MDPs with parity objectives provide the theoretical framework to study problems such as the verification and control of stochastic systems.

Qualitative and quantitative analysis. The analysis of MDPs with parity objectives can be classified into qualitative and quantitative analysis. Given an MDP with parity objective, the *qualitative analysis* asks for the computation of the set of vertices from where the parity objective can be ensured with probability 1 (almost-sure winning). The more general *quantitative analysis* asks for the computation of the maximal (or minimal) probability at each state with which the controller can satisfy the parity objective.

Importance of qualitative analysis. The qualitative analysis of MDPs is an important problem in verification that is of interest independent of the quantitative analysis problem. There are many applications where we need to know whether the correct behavior arises with probability 1. For instance, when analyzing a randomized embedded scheduler, we are interested in whether every thread progresses with probability 1 [9]. Even in settings where it suffices to satisfy certain specifications with probability $p < 1$, the correct choice of p is a challenging problem, due to the simplifications introduced during modeling. For example, in the analysis of randomized distributed algorithms it is quite common to require correctness with probability 1 (see, e.g., [16, 14, 19]). Furthermore, in contrast to quantitative analysis, qualitative analysis is robust to numerical perturbations and modeling errors in the transition probabilities, and consequently the algorithms for qualitative analysis are combinatorial. Finally, for MDPs with parity objectives, the best known algorithms and all algorithms used in practice first perform the qualitative analysis, and then perform a quantitative analysis on the result of the qualitative analysis [7, 8, 6]. Thus qualitative analysis for MDPs with parity objectives is one of the most fundamental and core problems in verification of probabilistic systems.

Previous results. The qualitative analysis for MDPs with parity objectives is achieved by iteratively applying solutions of the qualitative analysis of MDPs with Büchi objectives [7, 8, 6]. The qualitative analysis of an MDP with a parity objective with d priorities can be achieved by $O(d)$ calls to an algorithm for qualitative analysis of MDPs with Büchi objectives, and hence we focus on MDPs with Büchi objectives. The qualitative analysis problem for MDPs with Büchi objectives has been widely studied. The classical algorithm for the problem was given in [7, 8], and the worst case running time of the classical algorithm is $O(n \cdot m)$ time, where n is the number of vertices, and m is the number of edges of the MDP. Many improved algorithms have also been given in literature, such as [5, 3, 4], and the current best known worst case complexity of the problem is $O(\min\{n^2, m \cdot \sqrt{m}\})$. Moreover, there exists a family of MDPs where the running time of the improved algorithms match the above bound. While the worst case complexity of the problem has been studied, to the best of our knowledge the average case complexity of none of the algorithms has been studied in literature.

Our contribution. In this work we study for the first time the average case complexity of the qualitative analysis of MDPs with Büchi objectives. Specifically we study the average case complexity of the classical algorithm for the following two reasons: (1) the classical algorithm is very simple and appealing as it iteratively uses solutions of the standard graph reachability and alternating graph reachability algorithms, and can be implemented efficiently by symbolic algorithms; and (2) for the more involved improved algorithms it has been established that there are simple variants of the improved algorithms that never require more than an additional linear time as compared to the classical algorithm, and hence the average case complexity of these variants is no more than the average case complexity

of the classical algorithm. We study the average case complexity of the classical algorithm and establish that as compared to the quadratic worst case complexity, the average case complexity is linear. Our main contributions are summarized below:

1. *MDPs with constant out-degree.* We first consider MDPs with constant out-degree. In practice, MDPs often have constant out-degree: for example, see [10] for MDPs with large state space but constant number of actions, or [12, 17] for examples from inventory management where MDPs have constant number of actions (the number of actions correspond to the out-degree of MDPs). We consider MDPs where the out-degree of every vertex is fixed and given. The out-degree of a vertex v is d_v and there are constants $d_{\min}$ and $d_{\max}$ such that for every v we have $d_{\min} \leq d_v \leq d_{\max}$. Moreover, every subset of the set of vertices of size d_v is equally likely to be the neighbour set of v, independent of the neighbour sets of other vertices. We show that the expected number of iterations of the classical algorithm is at most logarithmic ($O(\log n)$), and the average case running time is linear ($O(n)$) (as compared to the worst case linear number of iterations and quadratic $O(n^2)$ time complexity of the classical algorithm, and the current best known $O(n \cdot \sqrt{n})$ worst case complexity). The average case complexity of this model implies the same average case complexity for several related models of MDPs with constant out-degree. For further discussion on this, see Remark 3.4 following Theorem 16.
2. *MDPs in the Erdös-Rényi model.* To consider the average case complexity over all MDPs, we consider MDPs where the underlying graph is a random directed graph according to the classical Erdös-Rényi random graph model [11]. We consider random graphs $\mathcal{G}_{n,p}$, over n vertices where each edge exists with probability p (independently of other edges). To analyze the average case complexity over all MDPs with all graphs equally likely we need to consider the $\mathcal{G}_{n,p}$ model with $p = \frac{1}{2}$ (i.e., each edge is present or absent with equal probability, and thus all graphs are considered equally likely). We show a stronger result (than only $p = \frac{1}{2}$) that if $p \geq \frac{c \cdot \log(n)}{n}$, for any constant $c > 2$, then the expected number of iterations of the classical algorithm is constant ($O(1)$), and the average case running time is linear (again as compared to the worst case linear number of iterations and quadratic time complexity). Note that we obtain that the average case (when $p = \frac{1}{2}$) running time for the classical algorithm is linear over all MDPs (with all graphs equally likely) as a special case of our results for $p \geq \frac{c \cdot \log(n)}{n}$, for any constant $c > 2$, since $\frac{1}{2} \geq \frac{3 \cdot \log(n)}{n}$ for $n \geq 17$. Moreover we show that when $p = \frac{1}{2}$ (i.e., all graphs are equally likely), the probability that the classical algorithm will require more than constantly many iterations is exponentially small (less than $\left(\frac{3}{4}\right)^n$).

Implications of our results. We now discuss several implications of our results. First, since we show that the classical algorithm has average case linear time complexity, it follows that the average case complexity of qualitative analysis of MDPs with Büchi objectives is linear time. Second, since qualitative analysis of MDPs with Büchi objectives is a more general problem than reachability in graphs (graphs are a special case of MDPs and reachability objectives are a special case of Büchi objectives), the best average case complexity that can be achieved is linear. Hence our results for the average case complexity are tight. Finally, since for the improved algorithms there are simple variants that never require more than linear time as compared to the classical algorithm it follows that the improved algorithms also have average case linear time complexity. Thus we complete the average case analysis of the algorithms for the qualitative analysis of MDPs with Büchi objectives. In summary our results show that the classical algorithm (the most simple and appealing algorithm) has excellent and optimal (linear-time) average case complexity as compared to the quadratic worst case complexity.

Technical contributions. The two key technical difficulties to establish our results are as follows: (1) Though there are many results for random undirected graphs, for the average case analysis of the classical algorithm we need to analyze random directed graphs; and (2) in contrast to other results related to random undirected graphs that prove results for almost all vertices, the classical algorithm stops when all vertices satisfy a certain reachability property; and hence we need to prove results for all vertices (as compared to almost all vertices). In this work we set up novel recurrence relations to estimate the expected number of iterations, and the average case running time of the classical algorithm. Our key technical results prove many interesting inequalities related to the recurrence relation for reachability properties of random directed graphs to establish the desired result. Detailed proofs omitted due to space restriction are available at: http://arxiv.org/abs/1202.4175.

2 Definitions

Markov decision processes (MDPs). A *Markov decision process (MDP)* $G = ((V, E), (V_1, V_P), \delta)$ consists of a directed graph (V, E), a partition (V_1, V_P) of the *finite* set V of vertices, and a probabilistic transition function δ: $V_P \to \mathcal{D}(V)$, where $\mathcal{D}(V)$ denotes the set of probability distributions over the vertex set V. The vertices in V_1 are the *player*-1 vertices, where player 1 decides the successor vertex, and the vertices in V_P are the *probabilistic (or random)* vertices, where the successor vertex is chosen according to the probabilistic transition function δ. We assume that for $u \in V_P$ and $v \in V$, we have $(u, v) \in E$ iff $\delta(u)(v) > 0$, and we often write $\delta(u, v)$ for $\delta(u)(v)$. For a vertex $v \in V$, we write $E(v)$ to denote the set $\{ u \in V \mid (v, u) \in E \}$ of possible out-neighbours, and $|E(v)|$ is the out-degree of v. For technical convenience we assume that every vertex in the graph (V, E) has at least one outgoing edge, i.e., $E(v) \neq \emptyset$ for all $v \in V$.

Plays, strategies and probability measure. An infinite path, or a *play*, of the game graph G is an infinite sequence $\omega = \langle v_0, v_1, v_2, \ldots \rangle$ of vertices such that $(v_k, v_{k+1}) \in E$ for all $k \in \mathbb{N}$. We write Ω for the set of all plays, and for a vertex $v \in V$, we write $\Omega_v \subseteq \Omega$ for the set of plays that start from the vertex v. A *strategy* for player 1 is a function σ: $V^* \cdot V_1 \to \mathcal{D}(V)$ that chooses the probability distribution over the successor vertices for all finite sequences $\vec{w} \in V^* \cdot V_1$ of vertices ending in a player-1 vertex (the sequence represents a prefix of a play). A strategy must respect the edge relation: for all $\vec{w} \in V^*$ and $u \in V_1$, if $\sigma(\vec{w} \cdot u)(v) > 0$, then $v \in E(u)$. Once a starting vertex $v \in V$ and a strategy $\sigma \in \Sigma$ is fixed, the outcome of the MDP is a random walk ω_v^σ for which the probabilities of events are uniquely defined, where an *event* $\mathcal{A} \subseteq \Omega$ is a measurable set of plays. For a vertex $v \in V$ and an event $\mathcal{A} \subseteq \Omega$, we write $\mathbb{P}_v^\sigma(\mathcal{A})$ for the probability that a play belongs to $\mathcal{A}$ if the game starts from the vertex v and player 1 follows the strategy σ.

Objectives. We specify *objectives* for the player 1 by providing a set of *winning* plays $\Phi \subseteq \Omega$. We say that a play ω *satisfies* the objective Φ if $\omega \in \Phi$. We consider *ω-regular objectives* [20], specified as parity conditions. We also consider the special case of Büchi objectives.

- *Büchi objectives.* Let B be a set of Büchi vertices. For a play $\omega = \langle v_0, v_1, \ldots \rangle \in \Omega$, we define $\mathrm{Inf}(\omega) = \{ v \in V \mid v_k = v \text{ for infinitely many } k \}$ to be the set of vertices that occur infinitely often in ω. The Büchi objectives require that some vertex of B be visited infinitely often, and defines the set of winning plays $\mathrm{Büchi}(B) = \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap B \neq \emptyset\}$.
- *Parity objectives.* For $c, d \in \mathbb{N}$, we write $[c..d] = \{ c, c+1, \ldots, d \}$. Let p: $V \to [0..d]$ be a function that assigns a *priority* $p(v)$ to every vertex $v \in V$, where $d \in \mathbb{N}$. The *parity objective* is defined as $\mathrm{Parity}(p) = \{ \omega \in \Omega \mid \min\big(p(\mathrm{Inf}(\omega))\big) \text{ is even } \}$. In other words,

the parity objective requires that the minimum priority visited infinitely often is even. In the sequel we will use Φ to denote parity objectives.

Qualitative analysis: almost-sure winning. Given a player-1 objective Φ, a strategy $\sigma \in \Sigma$ is *almost-sure winning* for player 1 from the vertex v if $\mathbb{P}_v^\sigma(\Phi) = 1$. The *almost-sure winning set* $\langle\langle 1 \rangle\rangle_{almost}(\Phi)$ for player 1 is the set of vertices from which player 1 has an almost-sure winning strategy. The qualitative analysis of MDPs correspond to the computation of the almost-sure winning set for a given objective Φ.

Algorithm for qualitative analysis. The almost-sure winning set for MDPs with parity objectives can be computed using $O(d)$ calls to compute the almost-sure winning set of MDPs with Büchi objectives [6, 7, 8]. Hence we focus on the qualitative analysis of MDPs with Büchi objectives. The algorithms for qualitative analysis for MDPs do not depend on the transition function, but only on the graph $G = ((V, E), (V_1, V_P))$. We now describe the classical algorithm for the qualitative analysis of MDPs with Büchi objectives and the algorithm requires the notion of random attractors.

Random attractor. Given an MDP G, let $U \subseteq V$ be a subset of vertices. The *random attractor* $Attr_P(U)$ is defined inductively as follows: $X_0 = U$, and for $i \geq 0$, let $X_{i+1} = X_i \cup \{ v \in V_P \mid E(v) \cap X_i \neq \emptyset \} \cup \{ v \in V_1 \mid E(v) \subseteq X_i \}$. In other words, X_{i+1} consists of (a) vertices in X_i, (b) probabilistic vertices that have at least one edge to X_i, and (c) player-1 vertices whose all successors are in X_i. Then $Attr_P(U) = \bigcup_{i \geq 0} X_i$. Observe that the random attractor is equivalent to the alternating reachability problem (reachability in AND-OR graphs).

Classical algorithm. The classical algorithm for MDPs with Büchi objectives is a simple iterative algorithm, and every iteration uses graph reachability and alternating graph reachability (random attractors). Let us denote the MDP in iteration i by G^i with vertex set V^i. Then in iteration i the algorithm executes the following steps: (i) computes the set Z^i of vertices that can reach the set of Büchi vertices $B \cap V^i$ in G^i; (ii) let $U^i = V^i \setminus Z^i$ be the set of remaining vertices; if U^i is empty, then the algorithm stops and outputs Z^i as the set of almost-sure winning vertices, and otherwise removes $Attr_P(U^i)$ from the graph, and continues to iteration $i+1$. The classical algorithm requires at most $O(n)$ iterations, where $n = |V|$, and each iteration requires at most $O(m)$ time, where $m = |E|$. Moreover the above analysis is tight, i.e., there exists a family of MDPs where the classical algorithm requires $\Omega(n)$ iterations, and total time $\Omega(n \cdot m)$. Hence $\Theta(n \cdot m)$ is the tight worst case complexity of the classical algorithm for MDPs with Büchi objectives. In this work we consider the average case analysis of the classical algorithm.

3 Average Case Analysis for MDPs with Constant Out-degree

In this section we consider the average case analysis of the number of iterations and the running time of the classical algorithm for computing the almost-sure winning set for MDPs with Büchi objectives on family of graphs with constant out-degree (out-degree of every vertex fixed and bounded by two constants $d_{\min}$ and $d_{\max}$).

Family of graphs and results. We consider families of graphs where the vertex set V ($|V| = n$), the target set of Büchi vertices B ($|B| = t$), and the out-degree d_v of each vertex v is fixed across the whole family. The only varying component is the edges of the graph; for each vertex v, every set of vertices of size d_v is equally likely to be the neighbour set of v, independent of neighbours of other vertices. Finally, there exist constants $d_{\min}$ and $d_{\max}$ such that $d_{\min} \leq d_v \leq d_{\max}$ for all vertices v. We will show the following for this family of graphs: (a) if the target set B has size more than $30 \cdot x \cdot \log(n)$, where x is the number

of distinct degrees, (i.e., $t \geq 30 \cdot x \cdot \log(n)$), then the expected number of iterations is $O(1)$ and the average running time is $O(n)$; and (b) if the target vertex set B has size at most $30 \cdot x \cdot \log(n)$, then the expected number of iterations required is at most $O(\log(n))$ and average running time is $O(n)$.

Notations. We use n and t for the total number of vertices and the size of the target set, respectively. We will denote by x the number of distinct out-degree d_v's, and let d_i, for $1 \leq i \leq x$ be the distinct out-degrees. Since for all vertices v we have $d_{\min} \leq d_v \leq d_{\max}$, it follows that we have $x \leq d_{\max} - d_{\min} + 1$. Let a_i be the number of vertices with degree d_i and t_i be the number of target (Büchi) vertices with degree d_i.

The event $R(k_1, k_2, ..., k_x)$. The *reverse reachable set* of the target set B is the set of vertices u such that there is a path in the graph from u to a vertex $v \in B$. Let S be any set comprising of k_i vertices of degree d_i, for $1 \leq i \leq x$. We define $R(k_1, k_2, ..., k_x)$ as the probability of the event that all vertices of S can reach B via a path that lies entirely in S. Due to symmetry between vertices, this probability only depends on k_i, for $1 \leq i \leq x$ and is independent of S itself. For ease of notation, we will sometimes denote the event itself by $R(k_1, k_2, ..., k_x)$. We will investigate the reverse reachable set of B, which contains B itself. Recall that t_i vertices in B have degree d_i, and hence we are interested in the case when $k_i \geq t_i$ for all $1 \leq i \leq x$.

Consider a set S of vertices that is the reverse reachable set, and let S be composed of k_i vertices of degree d_i and of size k, i.e., $k = |S| = \sum_{i=1}^{x} k_i$. Since S is the reverse reachable set, it follows that for all vertices v in $V \setminus S$, there is no edge from v to a vertex in S (otherwise there would be a path from v to a target vertex and then v would belong to S). Thus there are no incoming edges from $V \setminus S$ to S. Thus for each vertex v of $V \setminus S$, all its neighbours must lie in $V \setminus S$ itself. This happens with probability $\prod_{i \in [1,x], a_i \neq k_i} \left(\frac{\binom{n-k}{d_i}}{\binom{n}{d_i}} \right)^{a_i - k_i}$, since in $V \setminus S$ there are $a_i - k_i$ vertices with degree d_i and the size of $V \setminus S$ is $n - k$ (recall that $[1, x] = \{1, 2, \ldots, x\}$). Note that when $a_i \neq k_i$, there is at least one vertex of degree d_i in $V \setminus S$ that has all its neighbours in $V \setminus S$ and hence $n - k \geq d_i$. For simplicity of notation, we skip mentioning $a_i \neq k_i$ and substitute the term by 1 where $a_i = k_i$. The probability that each vertex in S can reach a target vertex is $R(k_1, k_2, ..., k_x)$. Hence the probability of S being the reverse reachable set is given by: $\prod_{i=1}^{x} \left(\frac{\binom{n-k}{d_i}}{\binom{n}{d_i}} \right)^{a_i - k_i} \cdot R(k_1, k_2, ..., k_x)$. There are $\prod_{i=1}^{x} \binom{a_i - t_i}{k_i - t_i}$ possible ways of choosing $k_i \geq t_i$ vertices (since the target set is contained) out of a_i. Notice that the terms are 1 where $a_i = k_i$. The value k can range from t to n and exactly one of these subsets of V will be the reverse reachable set. So the sum of probabilities of this happening is 1. Hence we have:

$$1 = \sum_{k=t}^{n} \sum_{\sum k_i = k, t_i \leq k_i \leq a_i} \left(\prod_{i=1}^{x} \binom{a_i - t_i}{k_i - t_i} \cdot \left(\frac{\binom{n-k}{d_i}}{\binom{n}{d_i}} \right)^{a_i - k_i} \right) \cdot R(k_1, k_2, ..., k_x) \tag{1}$$

Let

$$\begin{aligned} a_{k_1, k_2, \ldots, k_x} &= \left(\prod_{i=1}^{x} \binom{a_i - t_i}{k_i - t_i} \cdot \left(\frac{\binom{n-k}{d_i}}{\binom{n}{d_i}} \right)^{a_i - k_i} \right) \cdot R(k_1, k_2, ..., k_x); \\ \alpha_k &= \sum_{\sum k_i = k, t_i \leq k_i \leq a_i} a_{k_1, k_2, \ldots, k_x}. \end{aligned}$$

Our goal is to show that for $30 \cdot x \cdot \log(n) \leq k \leq n - 1$, the value of α_k is very small; i.e., we want to get an upper bound on α_k. Note that two important terms in α_k are

$\left(\binom{n-k}{d_i}/\binom{n}{d_i}\right)^{a_i-k_i}$ and $R(k_1, k_2, \ldots, k_x)$. Below we get an upper bound for both of them. Firstly note that when k is small, for any set S comprising of k_i vertices of degree d_i for $1 \leq i \leq x$ and $|S| = k$, the event $R(k_1, k_2, \ldots, k_x)$ requires each non-target vertex of S to have an edge inside S. Since k is small and all vertices have constant out-degree spread randomly over the entire graph, this is highly improbable. We formalize this intuitive argument in the following lemma.

▶ **Lemma 1** (Upper bound on $R(k_1, k_2, \ldots, k_x)$). *For $k \leq n - d_{\max}$*

$$R(k_1, k_2, \ldots, k_x) \leq \prod_{i=1}^{x} \left(1 - \left(1 - \frac{k}{n-d_i}\right)^{d_i}\right)^{k_i - t_i} \leq \prod_{i=1}^{x} \left(\frac{d_i \cdot k}{n - d_{\max}}\right)^{k_i - t_i}.$$

Now for $\left(\binom{n-k}{d_i}/\binom{n}{d_i}\right)^{a_i-k_i}$, we give an upper bound. First notice that when $a_i \neq k_i$, there is at least one vertex of degree d_i outside the reverse reachable set and it has all its edges outside the reverse reachable set. Hence, the size of the reverse reachable set (i.e. $n - k$) is at least d_i. Thus, $\binom{n-k}{d_i}$ is well defined.

▶ **Lemma 2.** *For any $1 \leq i \leq x$ such that $a_i \neq k_i$, we have* $\left(\frac{\binom{n-k}{d_i}}{\binom{n}{d_i}}\right)^{a_i-k_i} \leq \left(1 - \frac{k}{n}\right)^{d_i \cdot (a_i - k_i)}$.

Next we simplify the expression of α_k by taking care of the summation.

▶ **Lemma 3.** *The probability that the reverse reachable set is of size exactly k is α_k, and $\alpha_k \leq n^x \cdot \max_{\sum k_i = k, t_i \leq k_i \leq a_i} a_{k_1,k_2,\ldots,k_x}$.*

Now we proceed to achieve an upper bound on $a_{k_1,k_2,\ldots,k_x}$. First of all, intuitively if k is small, then $R(k_1, k_2, \ldots, k_x)$ is very small (this can be derived easily from Lemma 1). On the other hand, consider the case when k is very large. In this case there are very few vertices that cannot reach the target set. Hence they must have all their edges within them, which again has very low probability. Note that different factors that bind α_k depend on whether k is small or large. This suggests we should consider these cases separately. Our proof will consist of the following case analysis of the size k of the reverse reachable set: (1) when $30 \cdot x \cdot \log(n) \leq k \leq c_1 \cdot n$ is *small* (for some constant $c_1 > 0$); (2) when $c_1 \cdot n \leq k \leq c_2 \cdot n$ is *large* (for all constants $c_2 \geq c_1 > 0$); and (3) when $c_2 \cdot n \leq k \leq n - d_{\min} - 1$ is *very large*. The analysis of the constants will follow from the proofs. Note that since the target set B (with $|B| = t$) is a subset of its reverse reachable set, we have $k < t$ is infeasible. Hence in all the three cases, we will only consider $k \geq t$. We first consider the case when k is small.

3.1 Small k: $30 \cdot x \cdot \log(n) \leq k \leq c_1 n$

In this section we will consider the case when $30 \cdot x \cdot \log(n) \leq k \leq c_1 \cdot n$ for some constant $c_1 > 0$. Note that this case only occurs when $t \leq c_1 \cdot n$ (since $k \geq t$). We will assume this throughout this section. We will prove that there exists a constant $c_1 > 0$ such that for all $30 \cdot x \cdot \log(n) \leq k \leq c_1 \cdot n$ the probability (α_k) that the size of the reverse reachable set is k is bounded by $\frac{1}{n^2}$. Note that we already have a bound on α_k in terms of $a_{k_1,k_2,\ldots,k_x}$ (Lemma 3). We use continuous upper bounds of the discrete functions in $a_{k_1,k_2,\ldots,k_x}$ to convert it into a form that is easy to analyze. Let

$$b_{k_1,k_2,\ldots,k_x} = \prod_{i=1}^{x} \left(\frac{e \cdot (a_i - t_i)}{k_i - t_i}\right)^{k_i - t_i} \cdot e^{-\frac{k}{n} \cdot d_i \cdot (a_i - k_i)} \cdot \left(\frac{d_i \cdot k}{n - d_{\max}}\right)^{k_i - t_i}.$$

▶ **Lemma 4.** *We have* $a_{k_1,k_2,\ldots,k_x} \leq b_{k_1,k_2,\ldots,k_x}$.

Next we show that $b_{k_1,k_2,\ldots,k_x}$ drops exponentially as a function of k. This is the key and non-trivial result of this subsection and requires many involved mathematical inequalities. Note that this is the reason for the logarithmic lower bound on k in this section.

▶ **Lemma 5** (Upper bound on $b_{k_1,k_2,\ldots,k_x}$)**.** *There exists a constant* $c_1 > 0$ *such that for sufficiently large* n *and* $t \leq k \leq c_1 \cdot n$*, we have* $b_{k_1,k_2,\ldots,k_x} \leq \left(\frac{9}{10}\right)^k$.

Taking appropriate bounds on the value of k, we get an upper bound on $a_{k_1,k_2,\ldots,k_x}$. Recall that x is the number of distinct degrees and hence $x \leq d_{\max} - d_{\min} + 1$.

▶ **Lemma 6** (Upper bound on $a_{k_1,k_2,\ldots,k_x}$)**.** *There exists a constant* $c_1 > 0$ *such that for sufficiently large* n *with* $t \leq c_1 \cdot n$ *and for all* $30 \cdot x \cdot \log(n) \leq k \leq c_1 \cdot n$*, we have* $a_{k_1,k_2,\ldots,k_x} < \frac{1}{n^{3 \cdot x}}$.

▶ **Lemma 7** (Main lemma for small k)**.** *There exists a constant* $c_1 > 0$ *such that for sufficiently large* n *with* $t \leq c_1 \cdot n$ *and for all* $30 \cdot x \cdot \log(n) \leq k \leq c_1 \cdot n$*, the probability that the size of the reverse reachable set* S *is* k *is at most* $\frac{1}{n^2}$.

Proof. The probability that the reverse reachable set is of size k is given by α_k. By Lemma 3 and Lemma 6 we have α_k is at most $n^x \cdot n^{-3 \cdot x} = n^{-2 \cdot x} \leq \frac{1}{n^2}$. ■

3.2 Large k: $c_1 \cdot n \leq k \leq c_2 \cdot n$

In this section we will show that for all constants c_1 and c_2, with $0 < c_1 \leq c_2$, when $t \leq c_2 \cdot n$ the probability α_k is at most $\frac{1}{n^2}$ for all $c_1 \cdot n \leq k \leq c_2 \cdot n$. We start with a few notations. Let $a_i = p_i \cdot n, t_i = y_i \cdot n, k_i = s_i \cdot n$ for $1 \leq i \leq x$ and $k = s \cdot n$ for $c_1 \leq s < c_2$. We first present a bound on $a_{k_1,k_2,\ldots,k_x}$ in Lemma 8. In the following two lemmas we obtain an upper bound for the bound in Lemma 8. All the lemmas require to prove non-trivial mathematical inequalities to achieve the result.

▶ **Lemma 8.** *For all constants* c_1 *and* c_2 *with* $0 < c_1 \leq c_2$ *and for all* $c_1 \cdot n \leq k \leq c_2 \cdot n$*, we have* $a_{k_1,k_2,\ldots,k_x} \leq (n+1)^x \cdot \mathsf{Term}_1 \cdot \mathsf{Term}_2$*, where*

$$\mathsf{Term}_1 = \left(\prod_{i=1}^{x} \left(\frac{p_i - y_i}{s_i - y_i}\right)^{s_i - y_i} \left(\frac{p_i - y_i}{p_i - s_i}\right)^{p_i - s_i} (1-s)^{d_i(p_i - s_i)} (1 - (1-s)^{d_i})^{s_i - y_i} \right)^n, \quad \textit{and}$$

$$\mathsf{Term}_2 = \prod_{i=1}^{x} \left(\frac{1 - \left(1 - \frac{s}{1 - d_i/n}\right)^{d_i}}{1 - (1-s)^{d_i}} \right)^{n(s_i - y_i)}.$$

On simplification, the base of the exponent in Term_2 can be shown to be upper bounded by $1 + c^*/n$ for some constant $c^* > 0$. Since $s_i - y_i \leq 1$ and x is a constant, we have the following.

▶ **Lemma 9.** Term_2 *of Lemma 8 is upper bounded by a constant.*

For Term_1, we maximize the base of the exponent with respect to every d_i. When all d_i's take their optimal values, the value of the base becomes 1. But using the fact that $d_i \geq 2$ for all i, we show that not all the d_i's can take their optimal values simultaneously and we prove the following.

▶ **Lemma 10.** *There exists a constant $0 < \eta < 1$ such that* Term_1 *of Lemma 8 is at most* η^n.

▶ **Lemma 11** (Main lemma for large k)**.** *For all constants c_1 and c_2 with $0 < c_1 \leq c_2$, when n is sufficiently large and $t \leq c_2 \cdot n$, for all $c_1 \cdot n \leq k \leq c_2 \cdot n$, the probability that the size of the reverse reachable set S is k is at most $\frac{1}{n^2}$.*

Proof. By Lemma 8 we have $a_{k_1,k_2,\ldots,k_x} \leq (n+1)^x \cdot \mathsf{Term}_1 \cdot \mathsf{Term}_2$, and by Lemma 9 and Lemma 10 we have Term_2 is constant and Term_1 is exponentially small in n, where $x \leq (d_{\max} - d_{\min} + 1)$. The exponentially small Term_1 overrides the polynomial factor $(n+1)^x$ and the constant Term_2, and ensures that $a_{k_1,k_2,\ldots,k_x} \leq n^{-3x}$. By Lemma 3 it follows that $\alpha_k \leq n^{-2x} \leq \frac{1}{n^2}$. ■

3.3 Very large k: $(1 - 1/e^2)n$ to $n - d_{\min} - 1$

In this subsection we consider the case when the size k of the reverse reachable set is between $(1 - \frac{1}{e^2}) \cdot n$ and $n - d_{\min} - 1$. Note that if the reverse reachable set has size at least $n - d_{\min}$, then the reverse reachable set must be the set of all vertices, as otherwise the remaining vertices cannot have enough edges among themselves. Take $\ell = n - k$. Hence $d_{\min} + 1 \leq \ell \leq n/e^2$. As stated earlier, in this case $a_{k_1,k_2,\ldots,k_x}$ becomes small since we require that the ℓ vertices outside the reverse reachable set must have all their edges within themselves; this corresponds to the factor of $\left(\binom{n-k}{d_i}/\binom{n}{d_i}\right)^{a_i - k_i}$. Since ℓ is very small, this has a very low probability. With this intuition, we proceed to show the following bound on $a_{k_1,k_2,\ldots,k_x}$.

▶ **Lemma 12.** *We have $a_{k_1,k_2,\ldots,k_x} \leq \left(x \cdot e \cdot \frac{\ell}{n}\right)^{\ell}$.*

We see that $\left(x \cdot e \cdot \frac{\ell}{n}\right)^{\ell}$ is a convex function in ℓ and its maximum is attained at one of the endpoints. For $\ell = n/e^2$, the bound is exponentially decreasing with n where as for constant ℓ, the bound is polynomially decreasing in n. Hence, the maximum is attained at left endpoint of the interval (constant value of ℓ). However, the bound we get is not sufficient to apply Lemma 3 directly. An important observation is that as ℓ becomes smaller and smaller, the number of combinations $\sum k_i = k$, where $t_i \leq k_i \leq a_i$ in the expression of α_k also decrease. Thus, we break this case into two sub-cases.

▶ **Lemma 13.** *For $d_{\max} + 1 < \ell \leq n/e^2$, we have $a_{k_1,k_2,\ldots,k_x} < n^{-(2+x)}$ and $\alpha_k \leq 1/n^2$.*

▶ **Lemma 14.** *There exists a constant $h > 0$ such that for $d_{\min} + 1 \leq \ell \leq d_{\max} + 1$, we have $a_{k_1,k_2,\ldots,k_x} < h \cdot n^{-\ell}$ and $\alpha_k \leq \frac{h}{n^2}$.*

▶ **Lemma 15** (Main lemma for very large k)**.** *For all t, for all $(1 - \frac{1}{e^2}) \cdot n \leq k \leq n - 1$, the probability that the size of the reverse reachable set S is k is at most $O(\frac{1}{n^2})$.*

Proof. By Lemma 13 and Lemma 14 we obtain the result for all $(1 - \frac{1}{e^2}) \cdot n \leq k \leq n - d_{\min} - 1$. Since the reverse reachable set must contain all vertices if it has size at least $n - d_{\min}$, the result follows. ■

3.4 Expected Number of Iterations and Running Time

From Lemma 7, Lemma 11, and Lemma 15, we obtain that there exists a constant h such that (i) $\alpha_k \leq \frac{1}{n^2}$, for $30 \cdot x \cdot \log(n) \leq k < n - d_{\max} - 1$; (ii) $\alpha_k \leq \frac{h}{n^2}$, for $n - d_{\max} - 1 \leq k \leq n - d_{\min} - 1$; and (iii) $\alpha_k = 0$, for $n - d_{\min} \leq k \leq n - 1$. Hence using the union bound we

get the following result $\mathbb{P}(|S| < 30 \cdot x \cdot \log(n) \text{ or } |S| = n) \geq 1 - \frac{h}{n}$, where S is the reverse reachable set of target set (i.e., with probability at least $1 - \frac{h}{n}$ either at most $30 \cdot x \cdot \log(n)$ vertices reach the target set or all the vertices reach the target set). Let $I(n)$ and $T(n)$ denote the expected number of iterations and the expected running time of the classical algorithm for MDPs on random graphs with n vertices and constant out-degree. Then from above we have: $I(n) \leq \left(1 - \frac{h}{n}\right) \cdot 30 \cdot x \cdot \log(n) + \frac{h}{n} \cdot n$. It follows that $I(n) = O(\log(n))$. For the expected running time we have: $T(n) \leq \left(1 - \frac{h}{n}\right) \cdot (30 \cdot x \cdot \log(n))^2 + \frac{h}{n} \cdot n^2$. It follows that $T(n) = O(n)$. Hence we have the following theorem.

▶ **Theorem 16.** *The expected number of iterations and the expected running time of the classical algorithm for MDPs with Büchi objectives over graphs with constant out-degree are at most $O(\log(n))$ and $O(n)$, respectively.*

▶ Remark. For Theorem 16, we considered the model where the out-degree of each vertex v is fixed as d_v and there exist constants $d_{\min}$ and $d_{\max}$ such that $d_{\min} \leq d_v \leq d_{\max}$ for every vertex v. We discuss the implication of Theorem 16 for related models. First, when the out-degrees of all vertices are same and constant (say d^*), Theorem 16 can be applied with the special case of $d_{\min} = d_{\max} = d^*$. A second possible alternative model is when the outdegree of every vertex is a distribution over the range $[d_{\min}, d_{\max}]$. Since we proved that the average case is linear for every possible value of the outdegree d_v in $[d_{\min}, d_{\max}]$ for every vertex v (i.e., for all possible combinations), it implies that the average case is also linear when the outdegree is a distribution over $[d_{\min}, d_{\max}]$.

4 Average Case Analysis in Erdös-Rényi Model

In this section we consider the classical Erdös-Rényi model of random graphs $\mathcal{G}_{n,p}$, with n vertices, where each edge is chosen to be in the graph independently with probability p [11] (we consider directed graphs and then $\mathcal{G}_{n,p}$ is also referred as $\mathcal{D}_{n,p}$ in literature). First, in Section 4.1 we consider the case when p is $\Omega\left(\frac{\log(n)}{n}\right)$, and then we consider the case when $p = \frac{1}{2}$ (that generates the uniform distribution over all graphs). We will show two results: (1) if $p \geq \frac{c \cdot \log(n)}{n}$, for any constant $c > 2$, then the expected number of iterations is constant and the expected running time is linear; and (2) if $p = \frac{1}{2}$ (with $p = \frac{1}{2}$ we consider all graphs to be equally likely), then the probability that the number of iterations is more than one falls exponentially in n (in other words, graphs where the running time is more than linear are exponentially rare).

4.1 $\mathcal{G}_{n,p}$ with $p = \Omega\left(\frac{\log(n)}{n}\right)$

In this subsection we will show that given $p \geq \frac{c \cdot \log(n)}{n}$, for any constant $c > 2$, the probability that not all vertices can reach the given target set is at most $O(1/n)$. Hence the expected number of iterations of the classical algorithm for MDPs with Büchi objectives is constant and hence the algorithm works in average time linear in the size of the graph. Observe that to show the result the worst possible case is when the size of the target set is 1, as otherwise the chance that all vertices reach the target set is higher. Thus from here onwards, we assume that the target set has exactly 1 vertex.

The probability $R(n,p)$. For a random graph in $\mathcal{G}_{n,p}$ and a given target vertex, we denote by $R(n,p)$ the probability that each vertex in the graph has a path along the directed edges to the target vertex. Our goal is to obtain a lower bound on $R(n,p)$.

The key recurrence. Consider a random graph G with n vertices, with a given target vertex, and edge probability p. For a set K of vertices with size k (i.e., $|K| = k$), which contains the target vertex, $R(k,p)$ is the probability that each vertex in the set K, has a path to the target vertex, that lies within the set K (i.e., the path only visits vertices in K). The probability $R(k,p)$ depends only on k and p, due to the symmetry among vertices.

Consider the subset S of all vertices in V, which have a path to the target vertex. In that case, for all vertices v in $V \setminus S$, there is no edge going from v to a vertex in S (otherwise there would have been a path from v to the target vertex). Thus there are no incoming edges from $V \setminus S$ to S. Let $|S| = i$. Then the $i \cdot (n-i)$ edges from $V \setminus S$ to S should be absent, and each edge is absent with probability $(1-p)$. The probability that each vertex in S can reach the target is $R(i,p)$. So the probability of S being the reverse reachable set is given by:

$$(1-p)^{i\cdot(n-i)} \cdot R(i,p). \tag{2}$$

There are $\binom{n-1}{i-1}$ possible subsets of i vertices that include the given target vertex, and i can range from 1 to n. Exactly one subset S of V will be the reverse reachable set. So the sum of probabilities of the events that S is reverse reachable set is 1. Hence we have: $1 = \sum_{i=1}^{n} \binom{n-1}{i-1} \cdot (1-p)^{i\cdot(n-i)} \cdot R(i,p)$. Moving all but the last term (with $i = n$) to the other side, we get the following recurrence relation:

$$R(n,p) = 1 - \sum_{i=1}^{n-1} \binom{n-1}{i-1} \cdot (1-p)^{i\cdot(n-i)} \cdot R(i,p). \tag{3}$$

Bound on p for lower bound on $R(n,p)$. We will prove a lower bound on p in terms of n such that the probability that not all n vertices can reach the target vertex is less than $O(1/n)$. In other words, we require $R(n,p) \geq 1 - O\left(\frac{1}{n}\right)$. Since $R(i,p)$ is a probability value, it is at most 1. Hence from Equation 3 it follows that it suffices to show that

$$\sum_{i=1}^{n-1} \binom{n-1}{i-1} \cdot (1-p)^{i\cdot(n-i)} \cdot R(i,p) \leq \sum_{i=1}^{n-1} \binom{n-1}{i-1} \cdot (1-p)^{i\cdot(n-i)} \leq O\left(\frac{1}{n}\right) \tag{4}$$

to show that $R(n,p) \geq 1 - O\left(\frac{1}{n}\right)$. We will prove a lower bound on p for achieving Equation 4. Let us denote by $t_i = \binom{n-1}{i-1} \cdot (1-p)^{i\cdot(n-i)}$, for $1 \leq i \leq n-1$. The following lemma establishes a relation of t_i and t_{n-i}.

▶ **Lemma 17.** *For $1 \leq i \leq n-1$, we have $t_{n-i} = \frac{n-i}{i} \cdot t_i$.*

Define $g_i = t_i + t_{n-i}$, for $1 \leq i \leq \lfloor n/2 \rfloor$. From the previous lemma we have

$$g_i = t_{n-i} + t_i \quad = \quad \frac{n}{i} \cdot t_i = \frac{n}{i} \cdot \binom{n-1}{i-1} \cdot (1-p)^{i\cdot(n-i)} = \binom{n}{i} \cdot (1-p)^{i\cdot(n-i)}.$$

We observe that in the range of $[2, \lfloor \frac{n}{2} \rfloor]$, g_i attains its maximum value at one of the two endpoints. Then observing that $g_2 \leq t_1$ and $g_{\lfloor n/2 \rfloor} \leq t_1$, we conclude the following.

▶ **Lemma 18.** *For sufficiently large n, if $p \geq \frac{c\cdot\log(n)}{n}$ with $c > 2$, then $g_i \leq t_1$ for all $2 \leq i \leq \lfloor \frac{n}{2} \rfloor$.*

Now we simplify the expression of t_1 and prove the following using standard inequalities.

▶ **Lemma 19.** *For sufficiently large n, if $p \geq \frac{c\cdot\log(n)}{n}$ with $c > 2$, then $t_1 \leq \frac{1}{n^2}$.*

We are now ready to establish the main lemma that proves the upper bound on $R(n,p)$ and then the main result of the section.

▶ **Lemma 20.** *For sufficiently large n, for all $p \geq \frac{c\cdot\log(n)}{n}$ with $c > 2$, we have $R(n,p) \geq 1 - \frac{1.5}{n}$.*

▶ **Theorem 21.** *The expected number of iterations of the classical algorithm for MDPs with Büchi objectives for random graphs $\mathcal{G}_{n,p}$, with $p \geq \frac{c\cdot\log(n)}{n}$, where $c > 2$, is $O(1)$, and the average case running time is linear.*

Proof. By Lemma 20 it follows that $R(n,p) \geq 1 - \frac{1.5}{n}$, and if all vertices reach the target set, then the classical algorithm ends in one iteration. In the worst case the number of iterations of the classical algorithm is n. Hence the expected number of iterations is bounded by: $1 \cdot \left(1 - \frac{1.5}{n}\right) + n \cdot \frac{1.5}{n} = O(1)$. Since the expected number of iterations is $O(1)$ and every iteration takes linear time, it follows that the average case running time is linear. ■

4.2 Average-case analysis over all graphs

In this section, we consider uniform distribution over all graphs, i.e., all possible different graphs are equally likely. This is equivalent to considering the Erdös-Rényi model such that each edge has probability $\frac{1}{2}$. Using $\frac{1}{2} \geq 3 \cdot \log(n)/n$ (for $n \geq 17$) and the results from Section 4.1, we already know that the average case running time for $\mathcal{G}_{n,1/2}$ is linear. In this section we show that in $\mathcal{G}_{n,\frac{1}{2}}$, the probability that not all vertices reach the target is in fact exponentially small in n. It will follow that MDPs where the classical algorithm takes more than constant iterations are exponentially rare. We consider the same recurrence $R(n,p)$ as in the previous subsection and consider t_k and g_k as defined before. The following theorem shows the desired result.

▶ **Theorem 22.** *In $\mathcal{G}_{n,\frac{1}{2}}$ with sufficiently large n the probability that the classical algorithm takes more than one iteration is less than $\left(\frac{3}{4}\right)^n$.*

Proof. We first observe that Equation 3 and Equation 4 holds for all probabilities. Next we observe that Lemma 18 holds for $p \geq \frac{c\cdot\log(n)}{n}$ with any constant $c > 2$, and hence also for $p = \frac{1}{2}$ for sufficiently large n. We have $\sum_{i=1}^{n-1} t_i \leq \frac{3\cdot n}{2} \cdot t_1$. For $p = \frac{1}{2}$ we have $t_1 = \binom{n-1}{0} \cdot \left(1 - \frac{1}{2}\right)^{n-1} = \frac{1}{2^{n-1}}$. Hence we have $R(n,p) \geq 1 - \frac{3\cdot n}{2\cdot 2^{n-1}} > 1 - \frac{1.5^n}{2^n} = 1 - \left(\frac{3}{4}\right)^n$. The second inequality holds for sufficiently large n. It follows that the probability that the classical algorithm takes more than one iteration is less than $(\frac{3}{4})^n$. The desired result follows. ■

5 Conclusion

In this work both for the general case and the important special case of MDPs with constant out-degree we establish that the average case running time of the classical algorithm is linear, as compared to the quadratic worst case complexity. Moreover, as for the improved algorithms it is known that they require at most linear time more than the classical algorithm, it also follows that the average case running time of all the improved algorithms is also linear. We considered models where all MDPs in the relevant class are equally likely. We are not aware of any work that characterizes more appropriate probability distributions over graphs to represent MDPs that arise in practice. Characterizing distributions over MDPs that arise in practice and studying the average case complexity under such distributions is beyond the scope of this work, and is a subject for future work.

References

1. C. Baier and J-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
2. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *FSTTCS 95*, volume 1026 of *LNCS*, pages 499–513. Springer-Verlag, 1995.
3. K. Chatterjee and M. Henzinger. Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In *SODA*. ACM-SIAM, 2011.
4. K. Chatterjee and M. Henzinger. An $O(n^2)$ algorithm for alternating Büchi games. In *SODA*. ACM-SIAM, 2012.
5. K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Simple stochastic parity games. In *CSL'03*, volume 2803 of *LNCS*, pages 100–113. Springer, 2003.
6. K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Quantitative stochastic parity games. In *SODA'04*, pages 121–130. SIAM, 2004.
7. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
8. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.
9. L. de Alfaro, M. Faella, R. Majumdar, and V. Raman. Code-aware resource management. In *EMSOFT 05*. ACM, 2005.
10. L. de Alfaro and P. Roy. Magnifying-lens abstraction for markov decision processes. In *CAV*, pages 325–338, 2007.
11. P. Erdös and A. Rényi. On the evolution of random graphs. *Math. Inst. of the Hungarian Acad. of Sciences*, pages 17–61, 1960.
12. J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.
13. H. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
14. M. Kwiatkowska, G. Norman, and D. Parker. Verifying randomized distributed algorithms with prism. In *Workshop on Advances in Verification (WAVE'00)*, 2000.
15. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *TOOLS' 02*, pages 200–204. LNCS 2324, Springer, 2002.
16. A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. *Distributed Computing*, 13(3):155–186, 2000.
17. M. L. Puterman. *Markov Decision Processes*. J. Wiley and Sons, 1994.
18. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995. Technical Report MIT/LCS/TR-676.
19. M.I.A. Stoelinga. Fun with FireWire: Experiments with verifying the IEEE1394 root contention protocol. In *Formal Aspects of Computing*, 2002.
20. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 7, pages 389–455. Springer, 1997.

Verification of Open Interactive Markov Chains

Tomáš Brázdil[1], Holger Hermanns[2], Jan Krčál[1], Jan Křetínský[1,3], and Vojtěch Řehák[1]

1 **Faculty of Informatics, Masaryk University, Czech Republic**
{brazdil,krcal,jan.kretinsky,rehak}@fi.muni.cz
2 **Saarland University – Computer Science, Saarbrücken, Germany**
hermanns@cs.uni-saarland.de
3 **Institut für Informatik, Technical University Munich, Germany**

Abstract

Interactive Markov chains (IMC) are compositional behavioral models extending both labeled transition systems and continuous-time Markov chains. IMC pair modeling convenience - owed to compositionality properties - with effective verification algorithms and tools - owed to Markov properties. Thus far however, IMC verification did not consider compositionality properties, but considered closed systems. This paper discusses the evaluation of IMC in an open and thus compositional interpretation. For this we embed the IMC into a game that is played with the environment. We devise algorithms that enable us to derive bounds on reachability probabilities that are assured to hold in any composition context.

1998 ACM Subject Classification D.4.8 Performance

Keywords and phrases IMC, compositional verification, synthesis, time bounded reachability, discretization

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.474

1 Introduction

With the increasing complexity of systems and software reuse, component based development concepts gain more and more attention. In this setting developers are often facing the need to develop a component with only partial information about the surrounding components at hand, especially when relying on third-party components to be inter-operated with. This motivates verification approaches that ensure the functionality of a component in an environment whose behavior is unknown or only partially known. *Compositional verification* approaches aim at methods to prove guarantees on isolated components in such a way that when put together, the entire system's behavior has the desired properties based on the individual guarantees.

The assurance of reliable functioning of a system relates not only to its correctness, but also to its performance and dependability. This is a major concern especially in embedded system design. A natural instantiation of the general component-based approach in the continuous-time setting are *interactive Markov chains* [24]. Interactive Markov chains (IMC) are equipped with a sound compositional theory. IMC arise from classical labeled transition systems by incorporating the possibility to change state according to a random delay governed by some negative exponential distribution. This twists the model to one that is running in continuous real time. State transitions may be triggered by delay expirations, or may be triggered by the execution of actions. By dropping the new type of transitions, labeled transition systems are regained in their entirety. By dropping action-labeled transitions instead, one arrives at one of the simplest but also most widespread class of performance and de-

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 474–485

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

pendability models, *continuous-time Markov chains* (CTMC). IMC have a well-understood compositional theory, rooted in process algebra [3], and are in use as semantic backbones for dynamic fault trees [6], architectural description languages [5, 8], generalized stochastic Petri nets [25] and Statemate [4] extensions, and are applied in a large spectrum of practical applications, ranging from networked hardware on chips [15] to water treatment facilities [21] and ultra-modern satellite designs [16].

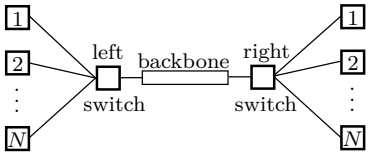
In recent years, various analysis techniques have been proposed [18, 27, 23, 26, 34, 19] for IMC. The pivotal verification problem considered is that of *time-bounded reachability*. It is the problem to calculate or approximate the probability that a given state (set) is reached within a given deadline. However, despite the fact that IMC support compositional model generation and minimization very well, the analysis techniques considered thus far are not compositional. They are all bound to the assumption that the analyzed IMC is closed, i.e. does not depend on interaction with the environment. Technically, this is related to the *maximal-progress assumption* governing the interplay of delay and action execution of an IMC component: Internal actions are assumed to happen instantaneously and therefore take precedence over delay transitions while external actions do not. External actions are the process algebraic means for interaction with other components. Remarkably, in all the published IMC verification approaches, all occurring actions are assumed to be internal (respectively internalized by means of a hiding operator prior to analysis).

In this paper, we instead consider *open* IMC, where the control over external actions is in the hands of and possibly delayed by an environment. The environment can be thought of as summarizing the behavior of one or several interacting components. As a consequence, we find ourselves in the setting of a timed game, where the environment has the (timed) control over external actions, while the IMC itself controls choices over internal actions. The resulting game turns out to be remarkably difficult, owed to the interplay of timed moves with external and internal moves of both players.

Concretely, assume we are given an IMC $\mathcal{C}$ which contains some internal non-deterministic transitions and also offers some external actions for synchronization to an unknown environment. Our goal is to synthesize a scheduler controlling the internal transitions which maximizes the probability of reaching a set G of goal states, in time T no matter what and when the environment E decides to synchronize with the external actions. The environment E ranges over all possible IMC able to synchronize with the external actions of $\mathcal{C}$.

To get a principal understanding of the complications faced, we need to consider a restricted setting, where $\mathcal{C}$ does not enable internal and external transitions at the same state. We provide an algorithm which approximates the probability in question up to a given precision $\varepsilon > 0$ and also computes an ε-optimal scheduler. The algorithm consists of two steps. First, we reduce the problem to a game where the environment is not an IMC but can decide to execute external actions at *non-deterministically* chosen time instances. In a second step, we solve the resulting game on $\mathcal{C}$ using discretization. Our discretization is based on the same approach as the algorithm of [34]. However, the algorithm as well as its proof of correctness is considerably more complicated due to presence of non-deterministic choices of the player controlling the environment. We finally discuss what happens if we allow internal and external transitions to be enabled at the same time.

Example. To illustrate the concepts by an example application, we can consider a variant of the *fault-tolerant workstation cluster* [22] depicted on the right. The overall system consists of two sub-clusters connected via a backbone; each of them contains N workstations. Any



component can fail and then needs to be repaired to become operational again. There is a single repair unit (not depicted) which must take decisions what to repair next when multiple components are failed. The entire system can be modelled using the IMC composition operators [22], but we are now also in the position to study a partial model, where some components, such as one of the switches, are left unspecified. We seek for the optimal repair schedule regardless of how the unknown components are implemented. We can answer questions such as: *"What is the worst case probability to hit a state in which premium service is not guaranteed within T time units?"* with premium service only being guaranteed if there are at least N operational workstations connected to each other via operational switches.

Our contribution. We investigate the problem of compositionally verifying open IMC. In particular, we introduce the problem of synthesizing optimal control for time-bounded reachability in an IMC interacting in an unknown environment, provided no state enables internal and external transition. Thereafter, we solve the problem of finding ε-optimal schedulers using the established method of discretization, give bounds on the size of the game to be solved for a given ε and thus establish upper complexity bound for the problem. Complete proofs and further relevant details can be found in the full version [10].

Related work. Model checking of *open* systems has been proposed in [28]. The synthesis problem is often stated as a *game* where the first player controls a component and the second player simulates an environment [31]. There is a large body of literature on games in verification, including recent surveys [1, 13]. *Stochastic* games have been applied to e.g. concurrent program synthesis [33] and for collaboration strategies among compositional stochastic systems [14]. Although most papers deal with discrete time games, lately games with stochastic *continuous-time* have gained attention [7, 30, 9, 11]. Some of the games we consider in the present paper exploit special cases of the games considered in [7, 11]. However, both papers prove decidability only for qualitative reachability problems and do not discuss compositionality issues. Further, while systems of [30, 9] are very similar to ours, the structure of the environment is fixed there and the verification is thus not compositional. The same holds for [32, 20], where time is under the control of the components.

The *time-bounded reachability* problem for closed IMC has been studied in [23, 34] and compositional abstraction techniques to compute it are developed in [26]. In the closed interpretation, IMC have some similarities with continuous-time Markov decision processes, CTMDP. Algorithms for time-bounded reachability in CTMDP and corresponding games are developed in [2, 9, 30]. A numerically stable algorithm for time-bounded properties for CTMDP is developed in [12].

2 Interactive Markov Chains

In this section, we introduce the formalism of interactive Markov chains together with the standard way to compose them. After giving the operational interpretation for closed systems, we define the fundamental problem of our interest, namely we define the value of time-bounded reachability and introduce the studied problems.

We denote by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{R}_{>0}$, and $\mathbb{R}_{\geq 0}$ the sets of natural numbers, natural numbers with zero, positive real numbers and non-negative real numbers, respectively.

▶ **Definition 1** (IMC). An interactive Markov chain (IMC) is a tuple $\mathcal{C} = (S, \mathbb{A}\mathrm{ct}^\tau, \hookrightarrow, \rightsquigarrow, s_0)$ where S is a finite set of *states*, $\mathbb{A}\mathrm{ct}^\tau$ is a finite set of *actions* containing a designated *internal action* τ, $s_0 \in S$ is an *initial* state,

- $\hookrightarrow \subseteq S \times \mathbb{A}\mathrm{ct}^\tau \times S$ is an *interactive transition* relation, and
- $\rightsquigarrow \subseteq S \times \mathbb{R}_{>0} \times S$ is a *Markovian transition* relation.

Elements of $\mathbb{A}\mathrm{ct} := \mathbb{A}\mathrm{ct}^\tau \setminus \{\tau\}$ are called *external actions*. We write $s \overset{a}{\hookrightarrow} t$ whenever $(s,a,t) \in \hookrightarrow$, and further $\mathrm{succ}_e(s) = \{t \in S \mid \exists a \in \mathbb{A}\mathrm{ct} : s \overset{a}{\hookrightarrow} t\}$ and $\mathrm{succ}_\tau(s) = \{t \in S \mid s \overset{\tau}{\hookrightarrow} t\}$. Similarly, we write $s \overset{\lambda}{\rightsquigarrow} t$ whenever $(s,\lambda,t) \in \rightsquigarrow$ where λ is called a *rate* of the transition, and $\mathrm{succ}_M(s) = \{t \in S \mid \exists \lambda : s \overset{\lambda}{\rightsquigarrow} t\}$. We assume w.l.o.g. that for each pair of states s and t, there is at most one Markovian transition from s to t. We say that an external, or internal, or Markovian transition is available in s if $\mathrm{succ}_e(s) \neq \emptyset$, or $\mathrm{succ}_\tau(s) \neq \emptyset$, or $\mathrm{succ}_M(s) \neq \emptyset$, respectively.

We also define a *total exit rate* function $\mathbf{E} : S \to \mathbb{R}_{\geq 0}$ which assigns to each state the sum of rates of all outgoing Markovian transitions, i.e. $\mathbf{E}(s) = \sum_{s \overset{\lambda}{\rightsquigarrow} t} \lambda$ where the sum is zero if $\mathrm{succ}_M(s)$ is empty. Furthermore, we define a probability matrix $\mathbf{P}(s,t) = \lambda/\mathbf{E}(s)$ if $s \overset{\lambda}{\rightsquigarrow} t$; and $\mathbf{P}(s,t) = 0$, otherwise.

IMC are well suited for compositional modeling, where systems are built out of smaller ones using composition operators. Parallel composition and hiding operators are central to the modeling style, where parallel components synchronize using shared action, and further synchronization can be prohibited by hiding (i.e. internalizing) some actions. IMC employ the *maximal progress assumption*: Internal actions take precedence over the advance of time [24].

▶ **Definition 2** (Parallel composition). For IMC $\mathcal{C}_1 = (S_1, \mathbb{A}\mathrm{ct}_1^\tau, \hookrightarrow_1, \rightsquigarrow_1, s_{01})$ and $\mathcal{C}_2 = (S_2, \mathbb{A}\mathrm{ct}_2^\tau, \hookrightarrow_2, \rightsquigarrow_2, s_{02})$ and a *synchronization alphabet* $A \subseteq \mathbb{A}\mathrm{ct}_1 \cap \mathbb{A}\mathrm{ct}_2$, the parallel composition $\mathcal{C}_1 \parallel_A \mathcal{C}_2$ is the IMC $\mathcal{C}_1 = (S_1 \times S_2, \mathbb{A}\mathrm{ct}_1^\tau \cup \mathbb{A}\mathrm{ct}_2^\tau, \hookrightarrow, \rightsquigarrow, (s_{01}, s_{02}))$ where $\hookrightarrow$ and $\rightsquigarrow$ are defined as the smallest relations satisfying

- $s_1 \overset{a}{\hookrightarrow} s_1'$ and $s_2 \overset{a}{\hookrightarrow} s_2'$ and $a \in A$ implies $(s_1,s_2) \overset{a}{\hookrightarrow} (s_1',s_2')$,
- $s_1 \overset{a}{\hookrightarrow} s_1'$ and $a \notin A$ implies $(s_1,s_2) \overset{a}{\hookrightarrow} (s_1',s_2)$ for each $s_2 \in S_2$,
- $s_2 \overset{a}{\hookrightarrow} s_2'$ and $a \notin A$ implies $(s_1,s_2) \overset{a}{\hookrightarrow} (s_1,s_2')$ for each $s_1 \in S_1$,
- $s_1 \overset{\lambda}{\rightsquigarrow} s_1'$ implies $(s_1,s_2) \overset{\lambda}{\rightsquigarrow} (s_1',s_2)$ for each $s_2 \in S_2$, and
- $s_2 \overset{\lambda}{\rightsquigarrow} s_2'$ implies $(s_1,s_2) \overset{\lambda}{\rightsquigarrow} (s_1,s_2')$ for each $s_1 \in S_1$.

▶ **Definition 3** (Hiding). For an IMC $\mathcal{C} = (S, \mathbb{A}\mathrm{ct}^\tau, \hookrightarrow, \rightsquigarrow, s_0)$ and a *hidden alphabet* $A \subseteq \mathbb{A}\mathrm{ct}$, the hiding $\mathcal{C} \setminus\!\!\setminus A$ is the IMC $(S, \mathbb{A}\mathrm{ct}^\tau \setminus A, \hookrightarrow', \rightsquigarrow, s_0)$ where $\hookrightarrow'$ is the smallest relation satisfying for each $s \overset{a}{\hookrightarrow} s'$ that $a \in A$ implies $s \overset{\tau}{\hookrightarrow}' s'$, and $a \notin A$ implies $s \overset{a}{\hookrightarrow}' s'$.

The analysis of IMC has thus far been restricted to *closed* IMC [18, 27, 23, 26, 34, 19]. In a closed IMC, external actions do not appear as transition labels (i.e. $\hookrightarrow \subseteq S \times \{\tau\} \times S$). In practice, this is achieved by an outermost hiding operator $\setminus\!\!\setminus \mathbb{A}\mathrm{ct}$ closing the composed system. Non-determinism among internal τ transitions is resolved using a (history-dependent) scheduler σ [34].

Let us fix a *closed* IMC $\mathcal{C} = (S, \mathbb{A}\mathrm{ct}^\tau, \hookrightarrow, \rightsquigarrow, s_0)$. The IMC $\mathcal{C}$ under a scheduler σ moves from state to state, and in every state may wait for a random time. This produces a *run* which is an infinite sequence of the form $s_0\, t_0\, s_1\, t_1 \cdots$ where s_n is the n-th visited state and t_n is the time spent there. After n steps, the scheduler resolves the non-determinism based on the *history* $\mathfrak{h} = s_0\, t_0 \cdots s_{n-1}\, t_{n-1}\, s_n$ as follows.

▶ **Definition 4** (Scheduler). A scheduler[1] for an IMC $\mathcal{C} = (S, \mathbb{A}\mathrm{ct}^\tau, \hookrightarrow, \rightsquigarrow, s_0)$ is a measurable[2] function $\sigma : (S \times \mathbb{R}_{\geq 0})^* \times S \to S$ such that for each history $\mathfrak{h} = s_0\, t_0\, s_1 \cdots s_n$ with $\mathrm{succ}_\tau(s_n) \neq \emptyset$ we have $\sigma(\mathfrak{h}) \in \mathrm{succ}_\tau(s_n)$. The set of all schedulers for $\mathcal{C}$ is denoted by $\mathfrak{S}(\mathcal{C})$.

[1] For the sake of simplicity, we only consider deterministic schedulers in this paper.
[2] More precisely, $\sigma^{-1}(s)$ is measurable in the product topology of the discrete topology on S and the Borel topology on $\mathbb{R}_{\geq 0}$.

The decision of the scheduler $\sigma(\mathfrak{h})$ determines t_n and s_{n+1} as follows. If $\mathrm{succ}_\tau(s_n) \neq \emptyset$, then the run proceeds immediately, i.e. in time $t_n := 0$, to the state $s_{n+1} := \sigma(\mathfrak{h})$. Otherwise, if $\mathrm{succ}_\tau(s_n) = \emptyset$, then only Markovian transitions are available in s_n. In such a case, the run moves to a randomly chosen next state s_{n+1} with probability $\mathbf{P}(s_n, s_{n+1})$ after waiting for a random time t_n chosen according to the exponential distribution with the rate $\mathbf{E}(s_n)$.

One of the fundamental problems in verification and performance analysis of continuous-time stochastic systems is the time-bounded reachability. Given a set of goal states $G \subseteq S$ and a time bound $T \in \mathbb{R}_{\geq 0}$, the *value of time-bounded reachability* is defined as $\sup_{\sigma \in \mathfrak{S}(\mathcal{C})} \mathcal{P}^\sigma_{\mathcal{C}}\left[\Diamond^{\leq T} G\right]$ where $\mathcal{P}^\sigma_{\mathcal{C}}\left[\Diamond^{\leq T} G\right]$ denotes the probability that a run of $\mathcal{C}$ under the scheduler σ visits a state of G before time T. The pivotal problem in the algorithmic analysis of IMC is to compute this value together with a scheduler that achieves the supremum. As the value is not rational in most cases, the aim is to provide an efficient approximation algorithm and compute an ε-optimal scheduler. The value of time-bounded reachability can be approximated up to a given error tolerance $\varepsilon > 0$ in time $\mathcal{O}(|S|^2 \cdot (\lambda T)^2/\varepsilon)$ [29], where λ is the maximal rate of $\mathcal{C}$, and the procedure also yields an ε-optimal scheduler. We generalize both the notion of the value as well as approximation algorithms to the setting of *open* IMC, i.e. those that are not closed, and motivate this extension in the next section.

3 Compositional Verification

In this section we turn our attention to the central questions studied in this paper. How can we decide how well an IMC component $\mathcal{C}$ performs (w.r.t. time-bounded reachability) when acting in parallel with an unknown environment? And how to control the component to establish a guarantee as high as possible?

Speaking thus far in vague terms, this amounts to finding a scheduler σ for $\mathcal{C}$ which maximizes the probability of reaching a target set G before T no matter what environment E is composed with $\mathcal{C}$. As we are interested in compositional modeling using IMC, the environments are supposed to be IMC with the same external actions as $\mathcal{C}$ (thus resolving the external non-determinism of $\mathcal{C}$). We also need to consider all resolutions of the internal non-determinism of E as well as the non-determinism arising from synchronization of $\mathcal{C}$ and E using another scheduler π. So we are interested in the following value:

$$\sup_{\sigma} \inf_{E,\pi} \mathcal{P}[G \text{ is reached in composition of } \mathcal{C} \text{ and } E \text{ before } T \text{ using } \sigma \text{ and } \pi].$$

Now, let us be more formal and fix an IMC $\mathcal{C} = (S, \mathbb{A}\mathrm{ct}^\tau, \hookrightarrow, \rightsquigarrow, s_0)$. For a given environment IMC E with the same action alphabet $\mathbb{A}\mathrm{ct}^\tau$, we introduce a composition

$$\mathcal{C}(E) = (\mathcal{C} \parallel_{\mathbb{A}\mathrm{ct}} E) \setminus \mathbb{A}\mathrm{ct}$$

where all open actions are hidden, yielding a closed system. Note that the states of $\mathcal{C}(E)$ are pairs (c, e) where c is a state of $\mathcal{C}$ and e is a state of E. We consider a scheduler σ of $\mathcal{C}$ and a scheduler π of $\mathcal{C}(E)$ respecting σ on internal actions of $\mathcal{C}$. We say that π *respects* σ, denoted by $\pi \in \mathfrak{S}(\mathcal{C}(E), \sigma)$, if for every history $\mathfrak{h} = (c_0, e_0)\, t_0 \cdots t_{n-1} (c_n, e_n)$ of $\mathcal{C}(E)$ the scheduler π satisfies one of the following conditions:

- $\pi(\mathfrak{h}) = (c, e)$ where $c_n \xhookrightarrow{a} c$ and $e_n \xhookrightarrow{a} e$ (π resolves synchronization)
- $\pi(\mathfrak{h}) = (c_n, e)$ where $e_n \xhookrightarrow{\tau} e$ (π chooses a move in the environment)
- $\pi(\mathfrak{h}) = (\sigma(\mathfrak{h}_{\mathcal{C}}), e_n)$ where $\mathfrak{h}_{\mathcal{C}} = c_0 t_0 \cdots t_{n-1} c_n$ (π chooses a move in $\mathcal{C}$ according to σ).

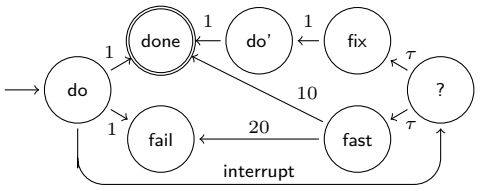
Given a set of goal states $G \subseteq S$ and a time bound $T \in \mathbb{R}_{\geq 0}$, the *value of compositional*

time-bounded reachability is defined as

$$\sup_{\sigma\in\mathfrak{S}(\mathcal{C})} \inf_{\substack{E\in\mathrm{ENV}\\ \pi\in\mathfrak{S}(\mathcal{C}(E),\sigma)}} \mathcal{P}^{\pi}_{\mathcal{C}(E)}\left[\Diamond^{\leq T} G_E\right] \qquad (*)$$

where ENV denotes the set of all IMC with the action alphabet $\mathbb{A}\mathrm{ct}^{\tau}$ and $G_E = G \times S_E$ where S_E is the set of states of E. As for the closed IMC, our goal is to efficiently approximate this value together with a maximizing scheduler. Before we present an approximation algorithm based on discretization, we illustrate some of the effects of the open system perspective.

Example. The figure on the right depicts an IMC on which we approximate the value $(*)$ for $T = 2$ and $G = \{\mathsf{done}\}$. From the initial state do, the system may go randomly either to the target done or to fail. Concurrently, the external action interrupt may switch the run to the state ?, where the scheduler σ chooses between two successors (1) the state fast allowing fast but risky run to the target and (2) the state fix that guarantees reaching the target but takes longer time. The value $(*)$ is approximately 0.47 and an optimal scheduler goes to fix only if there are more than 1.2 minutes left. Note that the probability of reaching the target in time depends on when the external action interrupt is taken. The most "adversarial" environment executes interrupt after 0.8 minutes from the start.



Results. We now formulate our main result concerning efficient approximation of the value of compositional time-bounded reachability. In fact, we provide an approximation algorithm for a restricted subclass of IMC defined by the following two assumptions:

▶ **Assumption 1.** *Each cycle contains a Markovian transition.*

This assumption is standard over all analysis techniques published for IMC [18, 27, 23, 26, 34, 19]. It implies that the probability of taking infinitely many transitions in finite time, i.e. of Zeno behavior, is zero. This is a rather natural assumption and does not restrict the modeling power much, since no real system will be able to take infinitely many transitions in finite time anyway. Furthermore, the assumed property is a compositional one, i.e. it is preserved by parallel composition and hiding.

▶ **Assumption 2.** *Internal and external actions are not enabled at the same time, i.e. for each state s, either* $\mathrm{succ}_e(s) = \emptyset$ *or* $\mathrm{succ}_\tau(s) = \emptyset$.

Note that both assumptions are met by the above mentioned example. However, Assumption 2 is not compositional; specifically, it is not preserved by applications of the hiding operator. A stronger assumption would require the environment not to trigger external actions in zero time after a state change. This is indeed implied by Assumption 2 which basically asks *internal* transitions of the component to be executed before any *external* actions are taken into account.[3] In fact, the reverse precedence cannot be implemented in real systems, if internal actions are assumed to be executed without delay. Any procedure implemented in $\mathcal{C}$ for checking the availability of external actions will involve some non-zero delay (unless one resorts to quantum effects). From a technical point of view, lifting Assumption 2 makes the studied problems considerably more involved; see Section 6 for further discussion.

[3] To see this one can construct a weak simulation relation between a system violating Assumption 2 and one satisfying it, where any state with both internal and external transitions is split into two: the first one enabling the internal transitions and a new τ to the second one only enabling the external ones.

▶ **Theorem 5.** *Let $\varepsilon > 0$ be an approximation bound and $\mathcal{C} = (S, \mathbb{A}\mathrm{ct}^\tau, \hookrightarrow, \rightsquigarrow, s_0)$ be an IMC satisfying Assumptions 1 and 2. Then one can approximate the value of compositional time-bounded reachability of $\mathcal{C}$ up to ε and compute an ε-optimal scheduler in time $\mathcal{O}(|S|^2 \cdot (\lambda T)^2/\varepsilon)$, where λ is the maximal rate of $\mathcal{C}$ and T is the reachability time-bound.*

In the remainder of the paper, we prove this theorem and discuss its restrictions. First, we introduce a new kind of real-time games, called CE games, that are played on open IMC. Then we reduce the compositional time-bounded reachability of $\mathcal{C}$ to time-bounded reachability objective in the CE game played just on the component $\mathcal{C}$ (see Proposition 6). In Section 5, we show how to reduce, using discretization, the time-bounded reachability in CE games to step-bounded reachability in discrete-time stochastic games (see Proposition 8), that in turn can be solved using simple backward propagation. Finally, we show, in Proposition 9, how to transform optimal strategies in the discretized stochastic games to ε-optimal schedulers for $\mathcal{C}$.

4 Game of Controller and Environment

In order to approximate $(*)$, the value of compositional time-bounded reachability, we turn the IMC $\mathcal{C}$ into a two-player *controller–environment game* (CE game) $\mathcal{G}$. The CE game naturally combines two approaches to real-time systems, namely the *stochastic* flow of time as present in CTMC with the *non-deterministic* flow of time as present in timed automata. The game $\mathcal{G}$ is played on the graph of an IMC $\mathcal{C}$ played by two players: **con** (controlling the component $\mathcal{C}$) and **env** (controlling/simulating the environment). In essence, **con** chooses in each state with internal transitions one of them, and **env** chooses in each state with external (and hence synchronizing) transitions either which of them should be taken, or a delay $t_e \in \mathbb{R}_{>0}$. Note that, due to Assumption 2, the players control the game in disjoint sets of states, hence $\mathcal{G}$ is a turn-based game. The internal and external transitions take zero time to be executed once chosen. If no zero time transition is chosen, the delay t_e determined by **env** competes with the Markovian transitions, i.e. with a random time sampled from the exponential distribution with the rate $\mathbf{E}(s)$. We consider time-bounded reachability objective, so the goal of **con** is to reach a given subset of states G before a given time T, and **env** opposes it.

Formally, let us fix an IMC $\mathcal{C} = (S, \mathbb{A}\mathrm{ct}^\tau, \hookrightarrow, \rightsquigarrow, s_0)$ and thus a CE game $\mathcal{G}$. A *run* of $\mathcal{G}$ is again an infinite sequence $s_0\, t_0\, s_1\, t_1 \cdots$ where $s_n \in S$ is the n-th visited state and $t_n \in \mathbb{R}_{\geq 0}$ is the time spent there. Based on the *history* $s_0\, t_0 \cdots t_{n-1}\, s_n$ went through so far, the players choose their moves as follows.

- If $\mathrm{succ}_\tau(s_n) \neq \emptyset$, the player **con** chooses a state $s_\tau \in \mathrm{succ}_\tau(s_n)$.
- Otherwise, the player **env** chooses either a state $s_e \in \mathrm{succ}_e(s_n)$, or a delay $t_e \in \mathbb{R}_{>0}$. (Note that if $\mathrm{succ}_e(s_n) = \emptyset$ only a delay can be chosen.)

Subsequently, Markovian transitions (if available) are resolved by randomly choosing a target state s_M according to the distribution $\mathbf{P}(s_n, \cdot)$ and randomly sampling a time t_M according to the exponential distribution with rate $\mathbf{E}(s_n)$. The next waiting time t_n and state s_{n+1} are given by the following rules in the order displayed.

- If $\mathrm{succ}_\tau(s_n) \neq \emptyset$ and s_τ was chosen, then $t_n = 0$ and $s_{n+1} = s_\tau$.
- If s_e was chosen, then $t_n = 0$ and $s_{n+1} = s_e$.
- If t_e was chosen then:
  - if $\mathrm{succ}_M(s_n) = \emptyset$, then $t_n = t_e$ and $s_{n+1} = s_n$;
  - if $t_e \leq t_M$, then $t_n = t_e$ and $s_{n+1} = s_n$;
  - if $t_M < t_e$, then $t_n = t_M$ and $s_{n+1} = s_M$.

According to the definition of schedulers in IMC, we formalize the choice of **con** as a *strategy* $\sigma : (S \times \mathbb{R}_{\geq 0})^* \times S \to S$ and the choice of **env** as a strategy $\pi : (S \times \mathbb{R}_{\geq 0})^* \times S \to S \cup \mathbb{R}_{>0}$. We denote by Σ and Π the sets of all strategies of the players **con** and **env**, respectively. In order to keep CE games out of Zeno behavior, we consider in Π only those strategies of the player **env** for which the induced Zeno runs have zero measure, i.e. the sum of the chosen delays diverges almost surely no matter what **con** is doing.

Given goal states $G \subseteq S$ and a time bound $T \in \mathbb{R}_{\geq 0}$, the *value of* $\mathcal{G}$ is defined as

$$\sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \mathcal{P}_{\mathcal{G}}^{\sigma,\pi}\left[\Diamond^{\leq T} G\right] \qquad (**)$$

where $\mathcal{P}_{\mathcal{G}}^{\sigma,\pi}\left[\Diamond^{\leq T} G\right]$ is the probability of all runs of $\mathcal{G}$ induced by σ and π and reaching a state of G before time T. We now show that the value of the CE game coincides with the value of compositional time-bounded reachability. This result is interesting and important as it allows us to replace unknown probabilistic behaviour with non-deterministic choices.

▶ **Proposition 6.** $(*) = (**)$, *i.e.*

$$\sup_{\sigma \in \mathfrak{S}(\mathcal{C})} \inf_{\substack{E \in \mathrm{ENV} \\ \pi \in \mathfrak{S}(\mathcal{C}(E),\sigma)}} \mathcal{P}_{\mathcal{C}(E)}^{\pi}\left[\Diamond^{\leq T} G_E\right] = \sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \mathcal{P}_{\mathcal{G}}^{\sigma,\pi}\left[\Diamond^{\leq T} G\right]$$

Proof Idea. We start with the inequality $(*) \geq (**)$. Let $\sigma \in \Sigma$ $(= \mathfrak{S}(\mathcal{C}))$ and let us fix an environment E together with a scheduler $\pi \in \mathfrak{S}(\mathcal{C}(E), \sigma)$. The crucial observation is that the purpose of the environment E (controlled by π) is to choose delays of external actions (the delay is determined by a sequence of internal and Markovian actions of E executed before the external action), which is in fact similar to the role of the player **env** in the CE game. The only difference is that the environment E "chooses" the delays randomly as opposed to deterministic strategies of **env**. However, using a technically involved argument, we show how to get rid of this randomization and obtain a strategy π' in the CE game satisfying $\mathcal{P}_{\mathcal{G}}^{\sigma,\pi'}\left[\Diamond^{\leq T} G\right] \leq \mathcal{P}_{\mathcal{C}(E)}^{\pi}\left[\Diamond^{\leq T} G_E\right]$.

Concerning the second inequality $(*) \leq (**)$, we show that every strategy of **env** can be (approximately) implemented using a suitable environment together with a scheduler π. The idea is to simulate every deterministic delay, say t, chosen by **env** using a random delay tightly concentrated around t (roughly corresponding to an Erlang distribution) that is implemented as an IMC. We show that the imprecision of delays introduced by this randomization induces only negligible alteration to the value. ◀

5 Discretization

In this section we show how to approximate the value $(**)$ of the CE game up to an arbitrarily small error $\varepsilon > 0$ by reduction to a discrete-time (turn-based) stochastic game Δ.

A stochastic game Δ is played on a graph $(V, \mapsto)$ partitioned into $V_\Box \uplus V_\Diamond \uplus V_\bigcirc$. A play starts in the initial vertex v_0 and forms a run $v_0 v_1 \cdots$ as follows. For a history $v_0 \cdots v_i$, the next vertex v_{i+1} satisfying $v_i \mapsto v_{i+1}$ is determined by a strategy $\sigma \in \Sigma_\Delta$ of player $\Box$ if $v_i \in V_\Box$ and by a strategy $\pi \in \Pi_\Delta$ of player $\Diamond$ if $v_i \in V_\Diamond$. Moreover, v_{i+1} is chosen randomly according to a fixed distribution $Prob(v_i)$ if $v_i \in V_\bigcirc$. For a formal definition, see, e.g., [17].

Let us fix a CE game $\mathcal{G}$ and a discretization step $\delta > 0$ that divides the time bound T into $N \in \mathbb{N}$ intervals of equal length (here $\delta = T/N$). We construct a discrete-time stochastic game Δ by substituting each state of $\mathcal{G}$ by a gadget of one or two vertices (as illustrated in Figure 1).[4] Intuitively, the game Δ models passing of time as follows. Each discrete step

[4] We assume w.l.o.g. that (1) states with internal transitions have no Markovian transitions available and

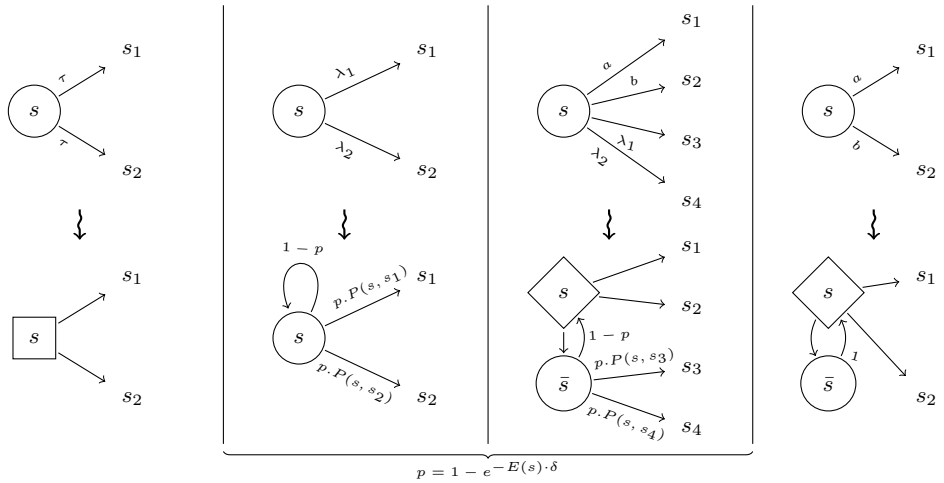


Figure 1 Four gadgets for transforming a CE game into a discrete game. The upper part shows types of states in the original CE game, the lower part shows corresponding gadgets in the transformed discrete game. In the lower part, the square-shaped, diamond-shaped and circle-shaped vertices belong to $V_\square$, $V_\Diamond$ and $V_\bigcirc$, respectively. Binary branching is displayed only in order to simplify the figure.

"takes" either time δ or time 0. Each step from a vertex of $V_\bigcirc$ takes time δ whereas each step from vertex of $V_\square \cup V_\Diamond$ takes zero time. The first gadget transforms internal transitions into edges of player $\square$ taking zero time. The second gadget transforms Markovian transitions into edges of player $\bigcirc$ taking time δ where the probability p is the probability that any Markovian transition is taken in $\mathcal{G}$ before time δ. The third gadget deals with states with both external and Markovian transitions available where the player $\Diamond$ decides in vertex s in zero time whether an external transition is taken or whether the Markovian transitions are awaited in $\overline{s}$ for time δ. The fourth gadget is similar, but no Markovian transition can occur and from $\overline{s}$ the play returns into s with probability 1.

Similarly to $(*)$ and $(**)$, we define the *value of the discrete-time game* Δ as

$$\sup_{\sigma \in \Sigma_\Delta} \inf_{\pi \in \Pi_\Delta} \mathcal{P}_\Delta^{\sigma,\pi}\left[\Diamond^{\#_\bigcirc \leq N} G\right] \qquad (***)$$

where $\mathcal{P}_\Delta^{\sigma,\pi}\left[\Diamond^{\#_\bigcirc \leq N} G\right]$ is the probability of all runs of Δ induced by σ and π that reach G before taking more than N steps from vertices in $V_\bigcirc$. According to the intuition above, such a step bound corresponds to a time bound $N \cdot \delta = T$.

We say that a strategy *is counting* if it only considers the last vertex and the current count $\#_\bigcirc$ of steps taken from vertices in $V_\bigcirc$. We may represent it as a function $V \times \{0, \ldots, N\} \to V$ since it is irrelevant what it does after more than N steps.

▶ **Lemma 7.** *There are counting strategies optimal in* $(***)$. *Moreover, they can be computed together with* $(***)$ *in time* $\mathcal{O}(N|V|^2)$.

We now show that the value $(***)$ of the discretized game Δ approximates the value $(**)$ of the CE game $\mathcal{G}$ and give the corresponding error bound.

(2) every state has at least one outgoing transition.This is no restriction since (1) Markovian transitions are never taken in such states and (2) any state without transitions can be endowed with a Markovian self-loop transition without changing the time-bounded reachability.

▶ **Proposition 8** (Error bound). *For every approximation bound $\varepsilon > 0$ and discretization step $\delta \leq \varepsilon/(\lambda^2 T)$ where $\lambda = \max_{s\in S} \mathbf{E}(s)$, the value $(***)$ induced by δ satisfies*

$$(***) \;\leq\; (**) \;\leq\; (***) + \varepsilon.$$

Proof Idea. The proof is inspired by the techniques for closed IMC [29]. Yet, there are several new issues to overcome, caused mainly by the fact that the player **env** in the CE game may choose an arbitrary real delay $t_e > 0$ (so **env** has uncountably many choices). The discretized game Δ is supposed to simulate the original CE game but restricts possible behaviors as follows: (1) Only one Markovian transition is allowed in any interval of length δ. (2) The delay t_e chosen by player $\Diamond$ (which simulates the player **env** from the CE game) must be divisible by δ. We show that none of these restrictions affects the value.

ad (1) As pointed out in [29], the probability of two or more Markovian transitions occurring in an interval $[0, \delta]$ is bounded by $(\lambda\delta)^2/2$ where $\lambda = \max_{s\in S} \mathbf{E}(s)$. Hence, the probability of multiple Markovian transitions occurring in any of the discrete steps of Δ is $\leq \varepsilon$.

ad (2) Assuming that at most one Markovian transition is taken in $[0, \delta]$ in the CE game, we reduce the decision when to take external transitions to minimization of a linear function on $[0, \delta]$, which in turn is minimized either in 0, or δ. Hence, the optimal choice for the player **env** in the CE game is either to take the transitions immediately at the beginning of the interval (before the potential Markovian transition) or to wait for time δ (after the potential Markovian transition). ◀

Finally, we show how to transform an optimal counting strategy $\sigma : V \times \{0, \ldots, N\} \to V$ in the discretized game Δ into an ε-optimal scheduler $\overline{\sigma}$ in the IMC $\mathcal{C}$. For every $\mathfrak{p} = s_0\, t_0 \cdots s_{n-1}\, t_{n-1}\, s_n$ we put $\overline{\sigma}(\mathfrak{p}) = \sigma(s_n, \lceil (t_0 + \ldots + t_{n-1})/\delta \rceil)$.

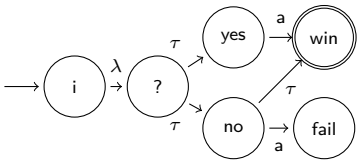
▶ **Proposition 9** (ε-optimal scheduler). *Let $\varepsilon > 0$, Δ be a corresponding discrete game, and $\overline{\sigma}$ be induced by an optimal counting strategy in Δ, then*

$$(*) \;\leq\; \inf_{\substack{E\in \mathrm{ENV} \\ \pi\in\mathfrak{S}(\mathcal{C}(E),\overline{\sigma})}} \mathcal{P}^{\pi}_{\mathcal{C}(E)}\left[\Diamond^{\leq T} G_E\right] + \varepsilon$$

This together with the complexity result of Lemma 7 finishes the proof of Theorem 5.

6 Summary, Discussion and Future Work

We discussed the computation of maximal timed bounded reachability for IMC operating in an unknown IMC environment to synchronize with. All prior analysis approaches considered closed systems, implicitly assuming that external actions do happen in zero time. Our analysis for open IMC works essentially with the opposite assumption, which is arguably more realistic. We have shown that the resulting stochastic two-player game has the same extremal values as a CE-game, where the player controlling the environment can choose exact times. The latter is approximated up to a given precision by discretization and the resulting control strategy translated back to a scheduler of the IMC achieving the bound.



Finally, we argue that lifting Assumption 2 makes analysis considerably more involved as the studied game may contain imperfect information and concurrent decisions. Let us illustrate the problems on an example. Consider an IMC depicted on the right. This IMC violates Assumption 2 in its state no. Let us fix an arbitrary environment E (controlled by π) and a scheduler σ. Since internal transitions of E take zero time, the environment must spend almost all the time in states without internal transitions. Hence, E is almost surely

in such a state when ? is entered. Assume E is in a state with the (external) action a being available. The scheduler σ wins if he chooses the internal transition to yes since the synchronizing transition a is then taken immediately, and fails if he chooses to proceed to no, as a (reasonable) scheduler π will now force synchronization on action a. If, otherwise, on entering state ?, E is in a state without the action a being available, the scheduler σ fails if he chooses yes because a (reasonable) environment never synchronizes, and wins if he chooses no since the environment E cannot immediately synchronize and the τ transition is taken. Note that the scheduler σ cannot observe whether a is available in the current state of E. As this is crucial for the further evolution of the game from state ?, the game is intrinsically of imperfect information.

We conjecture that solving even this special case of imperfect information games is PSPACE-hard. Yet, the complexity might only increase in the number of internal transitions that can be taken in a row. For systems, where a bound on the length of internal transition sequences can be assumed, this problem would then still be feasible.

Acknowledgement

The work has been supported by the Czech Science Foundation, grant No. P202/12/P612 (T. Brázdil, J. Křetínský and V. Řehák) and No. 102/09/H042 (J. Krčál) and by the DFG as part of the SFB/TR 14 AVACS, by the DFG/NWO project ROCKS, and by the EU FP7 project MEALS, grant agreement no. 295261. (H. Hermanns).

References

1 K. Apt and E. Grädel, editors. *Lectures in Game Theory for Computer Scientists*. Cambridge, 2011.

2 C. Baier, H. Hermanns, J.-P. Katoen, and B.R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theor. Comp. Sci.*, 345(1):2–26, 2005.

3 J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

4 E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, J. Rakow, R. Wimmer, and B. Becker. Compositional dependability evaluation for STATEMATE. *IEEE Trans. on Soft. Eng.*, 35(2):274–292, 2009.

5 H. Boudali, P. Crouzen, B.R. Haverkort, M. Kuntz, and M. I. A. Stoelinga. Architectural dependability evaluation with Arcade. In *Proc. of DSN*, pages 512–521. IEEE, 2008.

6 H. Boudali, P. Crouzen, and M. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *IEEE Trans. on DSC*, 7(2):128–143, 2010.

7 P. Bouyer and V. Forejt. Reachability in stochastic timed games. In *Proc. of ICALP*, volume 5556 of *LNCS*, pages 103–114. Springer, 2009.

8 M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. Safety, dependability and performance analysis of extended AADL models. *The Computer Journal*, 54(5):754–775, 2011.

9 T. Brázdil, V. Forejt, J. Krčál, J. Křetínský, and A. Kučera. Continuous-time stochastic games with time-bounded reachability. In *Proc. of FSTTCS*, volume 4 of *LIPIcs*, pages 61–72. Schloss Dagstuhl, 2009.

10 T. Brázdil, H. Hermanns, J. Krčál, J. Křetínský, and V. Řehák. Verification of open interactive markov chains. Technical Report FIMU-RS-2012-04, Faculty of Informatics MU, 2012.

11 T. Brázdil, J. Krčál, J. Křetínský, A. Kučera, and V. Řehák. Stochastic real-time games with qualitative timed automata objectives. In *Proc. of CONCUR*, volume 6269 of *LNCS*, pages 207–221. Springer, 2010.

12 P. Buchholz and I. Schulz. Numerical Analysis of Continuous Time Markov Decision processes over Finite Horizons. *Computers and Operations Research*, 38:651–659, 2011.

13 K. Chatterjee and T.A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

14 T. Chen, V. Forejt, M.Z. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. In *Proc. of TACAS*, volume 7214 of *LNCS*, pages 315–330. Springer, 2012.

15 N. Coste, H. Hermanns, E. Lantreibecq, and W. Serwe. Towards performance prediction of compositional models in industrial GALS designs. In *Proc. of CAV*, volume 5643, pages 204–218. Springer, 2009.

16 M.-A. Esteve, J.-P. Katoen, V.Y. Nguyen, B. Postma, and Y. Yushtein. Formal correctness, safety, dependability and performance analysis of a satellite. In *Proc. of ICSE*. ACM and IEEE press, 2012.

17 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1996.

18 H. Garavel, R. Mateescu, F. Lang, and W. Serwe. CADP 2006: A toolbox for the construction and analysis of distributed processes. In *Proc. of CAV*, volume 4590 of *LNCS*, pages 158–163. Springer, 2007.

19 D. Guck, T. Han, J.-P. Katoen, , and M.R. Neuhäußer. Quantitative timed analysis of interactive markov chains. In *NFM*, volume 7226 of *LNCS*, pages 8–23. Springer, 2012.

20 E.M. Hahn, G. Norman, D. Parker, B. Wachter, and L. Zhang. Game-based abstraction and controller synthesis for probabilistic hybrid systems. In *QEST*, pages 69–78, 2011.

21 B.R. Haverkort, M. Kuntz, A. Remke, S. Roolvink, and M.I.A. Stoelinga. Evaluating repair strategies for a water-treatment facility using Arcade. In *Proc. of DSN*, pages 419–424, 2010.

22 H. Hermanns and S. Johr. Uniformity by construction in the analysis of nondeterministic stochastic systems. In *DSN*, pages 718–728. IEEE Computer Society, 2007.

23 H. Hermanns and S. Johr. May we reach it? Or must we? In what time? With what probability? In *Proc. of MMB*, pages 125–140. VDE Verlag, 2008.

24 H. Hermanns and J.-P. Katoen. The how and why of interactive Markov chains. In *Proc. of FMCO*, volume 6286 of *LNCS*, pages 311–337. Springer, 2009.

25 H. Hermanns, J.-P. Katoen, M. R. Neuhäußer, and L. Zhang. GSPN model checking despite confusion. Technical report, RWTH Aachen University, 2010.

26 J.-P. Katoen, D. Klink, and M. R. Neuhäußer. Compositional abstraction for stochastic systems. In *Proc. of FORMATS*, volume 5813 of *LNCS*, pages 195–211. Springer, 2009.

27 J.-P. Katoen, I.S. Zapreev, E.M. Hahn, H. Hermanns, and D.N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.

28 O. Kupferman and M. Vardi. Module checking. In *CAV*, volume 1102 of *LNCS*, pages 75–86. Springer, 1996.

29 M.R. Neuhäußer. *Model checking nondeterministic and randomly timed systems*. PhD thesis, University of Twente, 2010.

30 M.N. Rabe and S. Schewe. Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. *Acta Informatica*, 48(5-6):291–315, 2011.

31 P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 1989.

32 J. Sproston. Discrete-time verification and control for probabilistic rectangular hybrid automata. In *QEST*, pages 79–88, 2011.

33 P. Černý, K. Chatterjee, T.A. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *CAV*, pages 243–259, 2011.

34 L. Zhang and M.R. Neuhäußer. Model checking interactive Markov chains. In *Proc. of TACAS*, volume 6015 of *LNCS*, pages 53–68. Springer, 2010.

On the Sensitivity of Shape Fitting Problems*

Kasturi Varadarajan[1] and Xin Xiao[2]

1 **Department of Computer Science**
University of Iowa, Iowa City, IA 52242, USA
kasturi-varadarajan@uiowa.edu

2 **Department of Computer Science**
University of Iowa, Iowa City, IA 52242, USA
xin-xiao@uiowa.edu

Abstract

In this article, we study shape fitting problems, ϵ-coresets, and total sensitivity. We focus on the (j,k)-projective clustering problems, including k-median/k-means, k-line clustering, j-subspace approximation, and the integer (j,k)-projective clustering problem. We derive upper bounds of total sensitivities for these problems, and obtain ϵ-coresets using these upper bounds. Using a dimension-reduction type argument, we are able to greatly simplify earlier results on total sensitivity for the k-median/k-means clustering problems, and obtain positively-weighted ϵ-coresets for several variants of the (j,k)-projective clustering problem. We also extend an earlier result on ϵ-coresets for the integer (j,k)-projective clustering problem in fixed dimension to the case of high dimension.

1998 ACM Subject Classification F.2.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Coresets, shape fitting, k-means, subspace approximation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.486

1 Introduction

In this article, we study shape fitting problem, coresets, and in particular, total sensitivity. A shape fitting problem is specified by a triple $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$, where $\mathbb{R}^d$ is the d-dimensional Euclidean space, $\mathcal{F}$ is a family of subsets of $\mathbb{R}^d$, and $\mathrm{dist} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ is a continuous function that we will refer to as a *distance* function. We also assume that (a) $\mathrm{dist}(p,q) = 0$ if and only if $p = q$, and (b) $\mathrm{dist}(p,q) = \mathrm{dist}(q,p)$. We refer to each $F \in \mathcal{F}$ as a *shape*, and we require each shape F to be a non-empty, closed, subset of $\mathbb{R}^d$. We define the *distance* of a point $p \in \mathbb{R}^d$ to a shape $F \in \mathcal{F}$ to be $\mathrm{dist}(p,F) = \min_{q \in F} \mathrm{dist}(p,q)$. An instance of a shape fitting problem is specified by a finite point set $P \subset \mathbb{R}^d$. We slightly abuse notation and use $\mathrm{dist}(P,F)$ to denote $\sum_{p \in P} \mathrm{dist}(p,F)$ when P is a set of points in $\mathbb{R}^d$. The goal is to find a shape which best fits P, that is, a shape minimizing $\sum_{p \in P} \mathrm{dist}(p,F)$ over all shapes $F \in \mathcal{F}$. This is referred to as the L_1 fitting problem, which is the main focus of this paper. In the L_∞ fitting problem, we seek to find a shape $F \in \mathcal{F}$ minimizing $\max_{p \in P} \mathrm{dist}(p,F)$.

In this paper, we focus on the (j,k)-projective clustering problem. Given non-negative integers j and k, the family of shapes is the set of k-tuples of affine j-subspaces (that is, j-flats) in $\mathbb{R}^d$. More precisely, each shape is the union of some k j-flats. The underlying distance function is usually the z^{th} power of the Euclidean distance, for a positive real number

* This material is based upon work supported by the National Science Foundation under Grant No. 0915543.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 486–497

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

z. When $j = 0$, $\mathcal{F}$ is the set of all k-point sets of $\mathbb{R}^d$, so the $(0, k)$-projective clustering problem is the k-median clustering problem when the distance function is the Euclidean distance, and it is the k-means clustering problem when the distance function is the square of the Euclidean distance; when $j = 1$, the family of shapes is the set of k-tuples of lines in $\mathbb{R}^d$; when $k = 1$, $(j, 1)$-projective clustering is the subspace approximation problem, where the family of shapes is the set of j-flats. Other than these projective clustering problems where j or k is set to specific values, another variant of the (j, k)-projective clustering problem is the integer (j, k)-projective clustering problem, where we assume that the input points have integer coordinates (but there is no restriction on j and k), and the magnitude of these coordinates is at most n^c, where n is the number of input points and $c > 0$ is some constant. That is, the points are in a polynomially large integer grid.

An ϵ-coreset for an instance P of a shape fitting problem is a weighted set S, such that for any shape $F \in \mathcal{F}$, the summation of distances from points in P approximates the weighted summation of the distances from points in S up to a multiplicative factor of $(1 \pm \epsilon)$. A more precise definition (Definition 1) follows later. Coresets can be considered as a succinct representation of the point set; in particular, in order to obtain a $(1 + \epsilon)$-approximation solution fitting P, it is sufficient to find a $(1 + \epsilon)$-approximation solution for the coreset S. One usually seeks a small coreset, whose *size* $|S|$ is independent of the cardinality of P. Coresets of size $o(n)$ for the (j, k)-projective clustering problem for general j and k are not known to exist. However, the k-median/k-means clustering, k-line clustering, j-subspace approximation, and integer (j, k)-projective clustering problems admit small coresets.

Langberg and Schulman [10] introduced a general approach to coresets via the notion of *sensitivity* of points in a point set, which provides a natural way to set up a probability distribution $\Pr\cdot$ on P. Roughly speaking, the sensitivity of a point with respect to a point set measures the importance of the point, in terms of fitting shapes in the given family of shapes $\mathcal{F}$. Formally, the sensitivity of point p in a point set P is defined by $\sigma_P(p) := \sup_{F\in\mathcal{F}} \mathrm{dist}(p, F)/\mathrm{dist}(P, F)$. (In the degenerate case where the denominator in the ratio is 0, the numerator is also 0, and we take the ratio to be 0; the reader should feel free to ignore this technicality.) The total sensitivity of a point set P is defined by $\mathfrak{S}_P := \sum_{p\in P} \sigma_P(p)$. The nice property of quantifying the "importance" of a point in a point set is that for any $F \in \mathcal{F}$, $\mathrm{dist}(p, F)/\mathrm{dist}(P, F) \le \sigma_P(p)$. Setting the probability of selecting p to be $\sigma_P(p)/\mathfrak{S}_P$, and the weight of p to be $\mathfrak{S}_P/\sigma_P(p)$, $\forall p \in P$, one can show that the variance of the sampling scheme is $O((\mathfrak{S}_P)^2)$. When $\mathfrak{S}_P$ is $o(n)$, (for example, a constant or logarithmic in terms of $n = |P|$), one can obtain an ϵ-coreset by sampling a small number of points. Langberg and Schulman [10] show that the total sensitivity of any (arbitrarily large) point set $P \subset \mathbb{R}^d$ for k-median/k-means clustering problem is a constant, depending only on k, independent of the cardinality of P and the dimension of the Euclidean space where P and $\mathcal{F}$ are from. Using this, they derived a coreset for these problems with size depending polynomially on d and k and independent of n. Their work can be seen as evolving from earlier work on coresets for the k-median/k-means and related problems via other low variance sampling schemes [3, 4, 7, 5].

Feldman and Langberg [6] relate the notion of an ϵ-coreset with the well-studied notion of an ϵ-approximation of range spaces. They use a "functional representation" of points: consider a family of functions $\mathcal{P} = \{f_p(\cdot)|p \in P\}$, where each point p is associated with a function $f_p : X \to \mathbb{R}$. The target here is to pick a small subset $S \subseteq P$ of points, and assign weights appropriately, so that $\sum_{p\in S} w_p f_p(x)$ approximates $\sum_{p\in P} f_p(x)$ at every $x \in X$. When X is $\mathcal{F}$ and $f_p(F) = \mathrm{dist}(p, F)$, this is just the original ϵ-coreset for P. However, $f_p(\cdot)$ can be any other function defined over $\mathcal{F}$, for example, $f_p(\cdot)$ can be the "residue distance"

of p, *i.e.,* $f_p(F) = |\text{dist}(p, F) - \text{dist}(p', F)|$, where p' is the projection of p on the optimum shape F^* fitting P. The definitions of sensitivities and total sensitivity easily carry over in this setting: $\sigma_{\mathcal{P}}(f_p) = \sup_{x \in X} f_p(x) / \sum_{f_q \in \mathcal{P}} f_q(x)$ (which coincides with $\sigma_P(p)$ when $f_p(\cdot)$ is $\text{dist}(p, \cdot)$), and $\mathfrak{S}_{\mathcal{P}} = \sum_{f_p \in \mathcal{P}} \sigma_{\mathcal{P}}(f_p)$ (which coincides with $\mathfrak{S}_P$ similarly). One of the results in [6] is that an approximating subset $S \subseteq P$ can be computed with the size $|S|$ upper bounded by the product of two quantities: $(\mathfrak{S}_{\mathcal{P}})^2$, and another parameter, the "dimension" (see Definition 3) of a certain range space induced by $\mathcal{P}$, denoted $\dim(\mathcal{P})$. We remark that $\dim(\mathcal{P})$ depends on d, which is the dimension of Euclidean space where P is from, and some other parameters related to X; when X is the family of shapes for the (j, k)-projective clustering problem, $\dim(\mathcal{P})$ also depends on j and k. This connection allows them to use many results from the well-studied area of ϵ-approximation of range spaces (such as deterministic construction of small ϵ-approximation of range spaces), thus constructing smaller coreset deterministically, and removes some routine analysis in the traditional way of obtaining coresets via random sampling.

1.1 Our Results

In this article, we prove upper bounds of total sensitivities for the (j, k)-projective clustering problems. In particular, we show a careful analysis of computing total sensitivities for shape fitting problems in high dimension. Total sensitivity $\mathfrak{S}_P$ for a point set $P \subset \mathbb{R}^d$ may depend on d: consider the shape fitting problem where the family of shapes is the set of hyperplanes, and P is a point set of size d in general position. Then clearly $\sigma_P(p) = 1$ (since there always exists a hyperplane containing all $d - 1$ points other than p), so $\mathfrak{S}_P = d$.

One question that arises naturally is that whether the dependence of the total sensitivity on the dimension d is essential. To answer this question, we show that if the distance function is Euclidean distance, or the z^{th} power of Euclidean distance for $z \in [1, \infty)$, then the total sensitivity function of a shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$ in the high dimensional space $\mathbb{R}^d$ is roughly the same as that of the low-dimensional variant $(\mathbb{R}^{d'}, \mathcal{F}', \text{dist})$, where d' is the "intrinsic" dimension of the shapes in $\mathcal{F}$, and $\mathcal{F}'$ consists of shapes contained in the low dimensional space $\mathbb{R}^{d'}$. A reification of this statement is that the total sensitivity function of the (j, k)-projective clustering is independent of d. For the (j, k)-projective clustering problems, the shapes are intrinsically low dimensional: each k-tuple of j-flats is contained in a subspace of dimension at most $k(j + 1)$. As we will see, the total sensitivity function for $(\mathbb{R}^d, \mathcal{F}, \text{dist})$, where $\mathcal{F}$ is the family of k-tuples of j-flats in $\mathbb{R}^d$, is of the same magnitude as the total sensitivity function of $(\mathbb{R}^{f(j,k)}, \mathcal{F}', \text{dist})$, where $f(j, k)$ is a function of j and k (which is independent of d), and $\mathcal{F}'$ is the family of k-tuples of j-flats in $\mathbb{R}^{f(j,k)}$.

We sketch our approach to upper bound the total sensitivity of the (j, k)-projective clustering. We first make the observation (Theorem 7 below) that the total sensitivity of a point set P is upper bounded by a constant multiple of the total sensitivity of $P' = \text{proj}(P, F^*)$, which is the projection of P on the optimum shape F^* fitting P in $\mathcal{F}$. The computation of total sensitivity of P' is very simple in certain cases; for example, for k-median clustering, P' is a multi-set which contains k distinct points, whose total sensitivity can be directly bounded by k. Therefore, we are able to greatly simplify the proofs in [10]. Another more important use of this observation is that it allows us to get a dimension-reduction type result for the (j, k)-projective clustering problems: note that although the point set and the shapes might be in a high dimension space $\mathbb{R}^d$, the projected point set P' lies in a

subspace of dimension $(j+1)k$ (since each k-tuple of j-flats is contained in a subspace of dimension at most $(j+1)k$), which is small under the assumption that both j and k are constant. Therefore, $\mathfrak{S}_P$, which usually depends on d if one directly computes it in a high dimensional space, depends only on j and k, since $\mathfrak{S}_P$ is $O(\mathfrak{S}_{P'})$.

Our method for bounding the total sensitivity directly translates into a template for computing ϵ-coresets:

1. Compute F^*, the optimal shape fitting P. (It suffices to use an approximately optimal shape.) Compute P', the projection of P onto F^*.
2. Compute a bound on the sensitivity of each point in P' with respect to P'. Since the ambient dimension is $O(jk)$, we may use a method that yields bounds on $\mathfrak{S}_{P'}$ with dependence on the ambient dimension. Use Theorem 7 to translate this into a bound for $\sigma_P(p)$ for each $p \in P$.
3. Sample points from P with probabilities proportional to $\sigma_P(p)$ to obtain a coreset, as described in [10, 6].

We now point out the difference between our usage of total sensitivity in the construction of coresets and the method in [6]. The construction of coresets in [6] may also be considered as based on total sensitivity, however in a very different way:

1. First obtain a small weighted point set $S \subseteq P$, such that $\mathrm{dist}(P,F) - \mathrm{dist}(P',F)$ is approximately the same as $\mathrm{dist}(S,F) - \mathrm{dist}(S',F)$ (S' is $\mathrm{proj}\,(S,F^*)$) for every $F \in \mathcal{F}$.
2. Then compute an ϵ-coreset $Q' \subseteq P'$ for the projected point set P', that is, $\mathrm{dist}(Q',F)$ approximates $\mathrm{dist}(P',F)$ for every $F \in \mathcal{F}$. (Since P' is from a low-dimensional subspace, the ambient dimension is small, and the computation can exploit this.)

Therefore, for each $F \in \mathcal{F}$, $\mathrm{dist}(P,F) = (\mathrm{dist}(P,F) - \mathrm{dist}(P',F)) + \mathrm{dist}(P',F) \approx (\mathrm{dist}(S,F) - \mathrm{dist}(S',F)) + \mathrm{dist}(Q',F)$.

Thus the weighted set $Q' \cup S \cup S'$ is a coreset for P, but notice that the points in S' have negative weights. In contrast, the weights of points in the coreset in our construction are positive. The advantage of getting coresets with positive weights is that in order to get an approximate solution to the shape fitting problem, we may run algorithms or heuristics developed for the shape fitting problem on the coreset, such as [1]. When points have negative weights, on the other hand, some of these heuristics do not work or need to be modified appropriately.

Another useful feature of the coresets obtained via our results is that the coreset is a subset of the original point set. When each point stands for a data item, the coreset inherits a natural interpretation. See [11] for a discussion of this issue in a broader context.

The sizes of the coresets in this paper are somewhat larger than the size of coresets in [6]. Roughly speaking, the size of the coreset in [6] is $f_1(d) + f_2(j,k)$, where $f_1(d)$ (respectively $f_2(j,k)$) is a function depending only on d (respectively j and k) for the (j,k)-projective clustering problem, while the coreset size in our paper is $f_1(d) \cdot f_2(j,k)$.

Organization of this paper: In this article, we focus on the construction that establishes small total sensitivity for various shape fitting problems, and the size of the resulting coreset. For clarity, we omit the description of algorithms for computing such bounds on sensitivity. Efficient algorithms result from the construction using a methodology that is now well-understood. Also because the weights for points in the coreset are nonnegative, the coreset lend itself to streaming settings, where points arrive one by one as $p_1, p_2, \cdots$ [9][6]. In Section 2, we present necessary definitions used through this article, and summarize related results from [6] and [12]. In Section 3, we prove the upper bound of total sensitivity of an instance of

a shape fitting problem in high dimension by its low dimensional projection. In Sections 4, 5, 6, and 7, we apply the upper bound from Section 3 to k-median/k-means, clustering, k-line clustering, j-subspace approximation, and the integer (j,k)-projective clustering problem, respectively, to obtain upper bounds for their total sensitivities, and the size of the resulting ϵ-coresets.

2 Preliminaries

In this section, we formally define some of the concepts studied in this article, and state crucial results from previous work. We begin by defining an ϵ-coreset.

▶ **Definition 1** (ϵ-coreset of a shape fitting problem)**.** Given an instance $P \subset \mathbb{R}^d$ of a shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$, and $\epsilon \in [0,1]$, an ϵ-coreset of P is a (weighted) set $S \subseteq P$, together with a weight function $w : S \to \mathbb{R}^+$, such that for any shape F in $\mathcal{F}$, it holds that $|\mathrm{dist}(P,F) - \mathrm{dist}(S,F)| \leq \epsilon \cdot \mathrm{dist}(P,F)$, where by definition, $\mathrm{dist}(P,F) = \sum_{p\in P} \mathrm{dist}(p,F)$, and $\mathrm{dist}(S,F) = \sum_{p\in S} w(p)\mathrm{dist}(p,F)$. The size of the weighted coreset S is defined to be $|S|$.

We note that in the literature, the requirement that the weights be non-negative, as well as the requirement that the coreset S be a subset of the original instance P, are sometimes relaxed. We include these requirements in the definition to emphasize that the coresets constructed here do satisfy them. We now define the sensitivities of points in a shape fitting instance, and the total sensitivity of the instance.

▶ **Definition 2** (Sensitivity of a shape fitting instance [10])**.** Given an instance $P \subset \mathbb{R}^d$ of a shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$, the sensitivity of a point p in P is $\sigma_P(p) := \inf\{\beta \geq 0 | \mathrm{dist}(p,F) \leq \beta \mathrm{dist}(P,F), \forall F \in \mathcal{F}\}$.

Note that an equivalent definition is to let $\sigma_P(p) = \sup_{F\in\mathcal{F}} \mathrm{dist}(p,F)/\mathrm{dist}(P,F)$, with the understanding that when the denominator in the ratio is 0, the ratio itself is 0.

The total sensitivity of the instance P, is defined by $\mathfrak{S}_P := \sum_{p\in P} \sigma_P(p)$. The total sensitivity function of the shape fitting problem is $\mathfrak{S}_n := \sup_{|P|=n} \mathfrak{S}_P$.

We now need a somewhat technical definition in order to be able to state an important earlier result from [6]. On a first reading, the reader is welcome to skip the detailed definition.

▶ **Definition 3** (The dimension of a shape fitting instance [6])**.** Let $P \subset \mathbb{R}^d$ be an instance of a shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$. For a weight function $w : P \to \mathbb{R}^+$, consider the set system $(P, \mathcal{R})$, where $\mathcal{R}$ is a family of subsets of P defined as follows: each element in $\mathcal{R}$ is a set of the form $R_{F,r}$ for some $F \in \mathcal{F}$ and $r \geq 0$, and $R_{F,r} = \{p \in P \mid w_p \cdot \mathrm{dist}(p,F) \leq r\}$. That is, $R_{F,r}$ is the set of those points in P whose weighted distance to the shape F is at most r. The dimension of the instance P of the shape fitting problem, denoted by $\dim(P)$, is the smallest integer m, such that for any weight function w and $A \subseteq P$ of size $|A| = a \geq 2$, we have: $|\{A \cap R_{F,r} | F \in \mathcal{F}, r \geq 0\}| \leq a^m$.

For instance, in the (j,k)-projective clustering problem with the underlying distance function dist being the z^{th} power of the Euclidean distance, the dimension $\dim(P)$ of any instance P is $O(jdk)$, independent of $|P|$ [6]. This is shown by methods similar to the ones used to bound the VC-dimension of geometric set systems. In fact, this bound is the only fact that we will need about the dimension of a shape fitting instance.

The following theorem recalls the connection established in [6] between coresets and sensitivity via the above notion of dimension.

▶ **Theorem 4** (Connection between total sensitivity and ϵ-coreset [6]). *Given any n-point instance $P \subset \mathbb{R}^d$ of a shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$, and any $\epsilon \in (0,1]$, there exists an ϵ-coreset for P of size $O\left(\left(\frac{\mathfrak{S}_n}{\epsilon}\right)^2 \dim(P)\right)$.*

Finally, we will need known bounds on the total sensitivity of (j,k)-projective clustering problem. These earlier bounds involve the dimension d corresponding to shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$.

▶ **Theorem 5** (Total sensitivity of (j,k)-projective clustering problem in fixed dimension [12]). *We have the following upper bounds of total sensitivities for the (j,k)-projective clustering problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$, where* dist *is the z-th power of the Eucldiean distance for $z \in (0,\infty)$.*

- *$j = 1$ (k-line center): $\mathfrak{S}_n$ is $O(k^{f(k,d)} \log n)$, where $f(d,k)$ is a function depending only on d and k.*
- *integer (j,k)-projective clustering problem: For any n-point instance P, with each coordinate being an integer of magnitude at most n^c for any constant $c > 0$, $\mathfrak{S}_P$ is $O((\log n)^{f(d,j,k)})$, where $f(d,j,k)$ is a function depending only on d, j, and k.*

3 Bounding the Total Sensitivity via Dimension Reduction

In this section, we show that the total sensitivity of a point set P is of the same order as that of $\text{proj}(P, F^*)$, which is the projection of P onto an optimum shape F^* from $\mathcal{F}$ fitting P. This result captures the fact that total sensitivity of a shape fitting problem quantifies the complexity of shapes, in the sense that total sensitivity depends on the dimension of smallest subspace containing each shape, regardless of the dimension of the ambient space where P is from.

▶ **Definition 6** (projection of points on a shape). For a shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$, define $\text{proj} : \mathbb{R}^d \times \mathcal{F} \to \mathbb{R}^d$, where $\text{proj}(p, F)$ is the projection of p on a shape F, that is, $\text{proj}(p, F)$ is a point in F which is nearest to p, with ties broken arbitrarily. That is, $\text{dist}(p, \text{proj}(p, F)) = \min_{q \in F} \text{dist}(p, q)$. We abuse the notation to denote the multi-set $\{\text{proj}(p, F) \,|\, p \in P\}$ by $\text{proj}(P, F)$ for $P \subset \mathbb{R}^d$.

We first show that $\mathfrak{S}_P$ is $O(\mathfrak{S}_{\text{proj}(P,F^*)})$, where F^* is an optimum shape fitting P from $\mathcal{F}$. In particular, this implies that when F^* is a low-dimensional object, the total sensitivity of $P \subset \mathbb{R}^d$ can be upper bounded by the total sensitivity of a point set contained in a low dimension subspace.

▶ **Theorem 7** (Dimension reduction, computing the total sensitivity of a point set in high dimensional space with the projected lower dimensional point set). *Given an instance P of a shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$, let F^* denote a shape that minimizes $dist(P, F)$ over all $F \in \mathcal{F}$. Let p' denote* proj (p, F^*) *and let P' denote* proj (P, F^*)*. Assume that the distance function satisfies the relaxed triangle inequality: $dist(p,q) \le \alpha(dist(p,r) + dist(r,q))$ for any $p, q, r \in \mathbb{R}^d$ for some constant $\alpha \ge 1$. Then*

1. *the following inequality holds: $\mathfrak{S}_P \le 2\alpha^2 \mathfrak{S}_{P'} + \alpha$.*
2. *if $dist(P, F^*) = 0$, then $\sigma_P(p) = \sigma_{P'}(p')$ for each $p \in P$. If $dist(P, F^*) > 0$, then $\sigma_P(p) \le \left(\alpha \frac{dist(p,p')}{dist(P,F^*)} + 2\alpha^2 \sigma_{P'}(p')\right)$.*

Proof. If $\text{dist}(P, F^*) = 0$, then $P = P'$, and clearly both parts of the theorem hold.

Let us consider the case where $\text{dist}(P, F^*) > 0$. By definition,

$$\sigma_P(p) = \inf\{\beta \ge 0 \mid \text{dist}(p, F) \le \beta \text{dist}(P, F), \forall F \in \mathcal{F}\},$$
$$\sigma_{P'}(p') = \inf\{\beta' \ge 0 \mid \text{dist}(p', F) \le \beta' \text{dist}(P', F), \forall F \in \mathcal{F}\}.$$

Let F be an arbitrary shape in $\mathcal{F}$. Then we have

$$\begin{aligned}
\mathrm{dist}(p,F) &\le \alpha\mathrm{dist}(p,p') + \alpha\mathrm{dist}(p',F) \\
&\le \alpha\mathrm{dist}(p,p') + \alpha\sigma_{P'}(p')\mathrm{dist}(P',F) \\
&\le \alpha\mathrm{dist}(p,p') + 2\alpha^2\sigma_{P'}(p')\mathrm{dist}(P,F) \\
&= \alpha\frac{\mathrm{dist}(p,p')}{\mathrm{dist}(P,F)}\cdot\mathrm{dist}(P,F) + 2\alpha^2\sigma_{P'}(p')\mathrm{dist}(P,F) \\
&\le \alpha\frac{\mathrm{dist}(p,p')}{\mathrm{dist}(P,F^*)}\cdot\mathrm{dist}(P,F) + 2\alpha^2\sigma_{P'}(p')\mathrm{dist}(P,F) \\
&= \left(\alpha\frac{\mathrm{dist}(p,p')}{\mathrm{dist}(P,F^*)} + 2\alpha^2\sigma_{P'}(p')\right)\mathrm{dist}(P,F).
\end{aligned}$$

The first inequality follows from the relaxed triangle inequality, the second inequality follows from the definition of sensitivity of p' in P', and third inequality follows from the fact that $\mathrm{dist}(P',F) = \sum_{p'\in P'}\mathrm{dist}(p',F) \le \sum_{p\in P}\alpha\left(\mathrm{dist}(p,F)+\mathrm{dist}(p,p')\right) = \alpha(\mathrm{dist}(P,F) + \mathrm{dist}(P,F^*)) \le 2\alpha\mathrm{dist}(P,F)$, since $\mathrm{dist}(P,F^*) \le \mathrm{dist}(P,F)$.

Thus the second part of the theorem holds. Now,

$$\begin{aligned}
\mathfrak{S}_P &= \sum_{p\in P}\sigma_P(p) \\
&\le \sum_{p\in P}\left(\alpha\frac{\mathrm{dist}(p,p')}{\mathrm{dist}(P,F^*)} + 2\alpha^2\sigma_{P'}(p')\right) \\
&= \alpha + 2\alpha^2\mathfrak{S}_{P'}.
\end{aligned}$$

◀

We make a remark regarding the value of α in Theorem 7 when the distance function is z^{th} power of Euclidean distance. It is used in Sections 4, 5, 6, and 7 when we derive upper bounds of total sensitivities for various shape fitting problems.

▶ Remark (Value of α when $\mathsf{dist}(\cdot,\cdot) = (\|\cdot\|_2)^z$). Let $z \in (0,\infty)$. Suppose $\mathrm{dist}(p,q) = (\|p-q\|_2)^z$. When $z \in (0,1)$, the weak triangle inequality holds with $\alpha = 1$; when $z \ge 1$, the weak triangle inequality holds with $\alpha = 2^{z-1}$. For a proof, see, for example, [8].

Theorem 7 bounds the total sensitivity of an instance P of a shape fitting problem $(\mathbb{R}^d,\mathcal{F},\mathrm{dist})$ in terms of the total sensitivity of P'. Suppose that there is an $m_2 \ll d$ so that each shape $F \in \mathcal{F}$ is in some subspace of dimension m_2. In the (j,k)-projective clustering problem, for example, $m_2 = k(j+1)$. Then note that P' is contained in a subspace of dimension m_2. Furthermore, when dist is the z^{th} power of the Euclidean distance, it turns out that for many shape fitting problems the sensitivity of P' can be bounded as if the shape fitting problem was housed in $\mathbb{R}^{2m_2}$ instead of $\mathbb{R}^d$. To see why this is the case for the (j,k)-projective clustering problem, fix an arbitrary subspace G of dimension $\min\{d,2m_2\}$ that contains P'. Then for for any $F \in \mathcal{F}$, there is an $F' \in \mathcal{F}$ such that (a) F' is contained in G, and (b) $\mathrm{dist}(p',F') = \mathrm{dist}(p',F)$ for all $p' \in P'$.

The following theorem summarizes this phenomenon. For simplicity, it is stated for the (j,k)-projective clustering problem, even though the phenomenon itself is somewhat more general.

▶ **Theorem 8** (Sensitivity of a lower dimensional point set in a high dimensional space)**.** *Let P' be an n-point instance of the (j,k)-projective clustering problem $(\mathbb{R}^d,\mathcal{F},\mathrm{dist})$, where* dist *is the z^{th} power of the Euclidean distance, for some $z \in (0,\infty)$. Assume that P' is*

contained in a subspace of dimension m_1. (Note that for each shape $F \in \mathcal{F}$, there is a subspace of dimension $m_2 = k(j+1)$ containing it.) Let G be any subspace of dimension $m = \min\{m_1 + m_2, d\}$ containing P'; fix an orthonormal basis for G, and for each $p' \in P'$, let $p'' \in \mathbb{R}^m$ be the coordinates of p' in terms of this basis. Let $P'' = \{p'' \mid p' \in P'\}$, and view P'' as an instance of the (j,k)-projective clustering problem $(\mathbb{R}^m, \mathcal{F}', \text{dist})$, where $\mathcal{F}'$ is the set of all k-tuples of j-subspaces in $\mathbb{R}^m$, and dist *is the z^{th} power of the Eucldiean distance. Then, $\sigma_{P'}(p') = \sigma_{P''}(p'')$ for each $p' \in P'$, and $\mathfrak{S}_{P'} = \mathfrak{S}_{P''}$.*

4 k-median/k-means Clustering Problem

In this section, we derive upper bounds for the total sensitivity function for the k-median/k-means problems, and its generalizations, where the distance function is z^{th} power of Euclidean distance, using the approach in Section 4. These bounds are similar to the ones derived by Langberg and Schulman [10], but the proof is much simplified. For the rest of the article, dist is assumed to be the z^{th} power of the Euclidean distance.

▶ **Theorem 9** (Total sensitivity of $(0,k)$-projective clustering)**.** *Consider the shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$, where $\mathcal{F}$ is the set of all k-point subsets of $\mathbb{R}^d$. We have the following upper bound on the total sensitivity:*

$$\mathfrak{S}_n \leq 2^{2z-1}k + 2^{z-1}, \qquad z \geq 1,$$
$$\mathfrak{S}_n \leq 2k + 1, \qquad z \in (0,1).$$

In particular, the total sensitivity of the k-median problem (which corresponds to the case when $z = 1$) is at most $2k+1$, and the total sensitivity of the k-means problem (which corresponds to the case when $z = 2$) is $8k + 2$.

Proof. Let P be an arbitrary n-point set. Apply Theorem 7, and note that $\text{proj}(P, C^*)$, where C^* is an optimum set of k centers, contains at most k distinct points. Assume that $C^* = \{c_1^*, c_2^*, \cdots, c_k^*\}$. Let P_i be the set of points in P whose projection is c_i^*, that is, $P_i = \{p \in P | \text{proj}(p, C^*) = c_i^*\}$. It is easy to see that the summation of sensitivities of the $|P_i|$ copies of c_i^* is at most 1: for any k-point set C in $\mathbb{R}^d$, $|P_i| \cdot \frac{\text{dist}(c_i^*, C)}{\text{dist}(C^*, C)} = \frac{|P_i|\text{dist}(c_i^*, C)}{\sum_{j=1}^{k} |P_j|\text{dist}(c_j^*, C)} \leq 1$.

Therefore, the total sensitivity of $\text{proj}(P, C^*)$ is at most k. Substituting α from the remark after Theorem 7, we get the above result. ◀

▶ **Theorem 10** (ϵ-coreset for $(0,k)$-projective clustering)**.** *Consider the shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$, where $\mathcal{F}$ is the set of all k-point subsets of $\mathbb{R}^d$. For any n-point instance P, there is an ϵ-coreset of size $O(k^3 d\epsilon^{-2})$.*

Proof. Observe that the $\dim(P)$ is $O(kd)$. Using Theorem 4, and Theorem 9, we obtain the above result. ◀

5 k-line Clustering Problem

In this section, we derive upper bounds on the total sensitivity function for the k-line clustering problem, that is, the $(1,k)$-projective clustering problem.

▶ **Theorem 11** (Total sensitivity for k-line clustering problem)**.** *Consider the shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$, where $\mathcal{F}$ is the set of k-tuple of lines. The total sensitivity function, $\mathfrak{S}_n$, is $O(k^{f(k)} \log n)$, where $f(k)$ is a function the depends only on k.*

Proof. Let P be an arbitrary n-point set. Let K^* denote an optimum set of k lines fitting P. Using Theorems 7 and 8, it suffices to bound the sensitivity of an n-point instance of a k-line clustering problem housed in $\mathbb{R}^{4k}$. By Theorem 5, the total sensitivity of this latter shape fitting problem is $O(k^{f(k)} \log n)$, where $f(k)$ is a function depending only on k. Therefore, $\mathfrak{S}_n$ is $O(k^{f(k)} \log n)$.

(Alternatively, one could use a recent result in [8]. Let P' denote the projection of P into K^*. Since K^* is a union of k lines, we can upper bound the sensitivity of P' by k times the sensitivity of an n-point set that lies on a *single line*. The sensitivity of an n-point set that lies on a single line can be upper bounded by the sensitivity of an n-point set for the *weighted* $(0, k)$-projective clustering problem, for which the sensitivity bound is $O(k^{f(k)} \log n)$ as shown in [8].) ◀

Notice that for k-line clustering problem, the bound on the total sensitivity depends logarithmically on n. We give below a construction of a point set that shows that this is necessary, even for $d = 2$.

▶ **Theorem 12** (The upper bound of total sensitivity for k-line clustering problem is tight)**.** *For every $n \geq 2$, there exists an n-point instance of the k-line clustering problem $(\mathbb{R}^2, \mathcal{F}, \mathrm{dist})$, where* dist *is the Euclidean distance, such that the total sensitivity of P is $\Omega(\log n)$.*

Proof. We construct a point set P of size n, together with n shapes $F_i \in \mathcal{F}$, $i = 1, \cdots, n$, such that $\sum_{i=1}^{n} \mathrm{dist}(p_i, F_i)/\mathrm{dist}(P, F_i)$ is $\Omega(\log n)$. Note that this implies that $\mathfrak{S}_P$ is at least $\Omega(\log n)$. Let P be the following point set in $\mathbb{R}^2$: $p_i = (1/2^{i-1}, 0)$, for $i = 1, \cdots, n$. Let F_i be a pair of lines: one vertical line and one horizontal line, where the vertical line is the y-axis, and the horizontal line is $\{(x, 1/2^i) | x \in \mathbb{R}\}$.

Consider the point p_i, where $i = 1, \cdots, n$. We show that $\mathrm{dist}(p_i, F_i)/\mathrm{dist}(P, F_i)$ is at least $1/(2 + i)$, for $i = 1, \cdots, n$. For $j \leq i$, note that $\mathrm{dist}(p_j, F_i) = 1/2^i$: since the distance from p_j to the horizontal line in F_i is $1/2^i$ and the distance to the vertical line is $1/2^{j-1}$, $\mathrm{dist}(p_j, F_i) = \min\{1/2^{j-1}, 1/2^i\} = 1/2^i$. For $i + 1 \leq j \leq n$, on the other hand, $\mathrm{dist}(p_j, F_i) = 1/2^{j-1}$. Therefore, $\sum_{j=i+1}^{n} \mathrm{dist}(p_j, F_i) = \sum_{j=i+1}^{n} 1/2^{j-1} = (1/2^{i-1}) \cdot (1 - (1/2)^{n-i})$. Thus, we have

$$\sigma_P(p_i) = \sup_{F \in \mathcal{F}} \frac{\mathrm{dist}(p_i, F)}{\mathrm{dist}(P, F)} \geq \frac{\mathrm{dist}(p_i, F_i)}{\mathrm{dist}(P, F_i)} = \frac{1/2^i}{(1/2^{i-1} - 1/2^{n-1}) + i \cdot (1/2^i)} > \frac{1}{2 + i}$$

Therefore, $\mathfrak{S}_P \geq \sum_{i=1}^{n} \sigma_P(p_i) > \sum_{i=1}^{n} \frac{1}{2+i}$, which is $\Omega(\log n)$. ◀

▶ **Theorem 13** (ϵ-coreset for k-line clustering problem)**.** *Consider the shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$, where $\mathcal{F}$ is the set of all k-tuples of lines in $\mathbb{R}^d$. For any n-point instance P, there is an ϵ-coreset with size $O(k^{f(k)} d (\log n)^2 / \epsilon^2)$.*

Proof. This result follows from Theorem 11, Theorem 4, and the fact that $\dim(P)$ in this case is $O(kd)$. ◀

6 Subspace approximation

In this section, we derive upper bounds on the sensitivity of the subspace approximation problem, that is, the $(j, 1)$-projective clustering problem. For the applications of Theorems 7 and 8 in the other sections, we use existing bounds on the sensitivity that have a dependence on the dimension d. For the subspace approximation problem, however, we derive here the dimension-dependent bounds on sensitivity by generalizing an argument from [10] for the

case $j = d-1$ and $z = 2$. This derivation is somewhat technical. With these bounds in hand, the derivation of the dimension-independent bounds is readily accomplished in a manner similar to the other sections.

6.1 Dimension-dependent bounds on Sensitivity

We first recall the notion of an (α, β, z)-*conditioned basis* from [5], and state one of its properties (Lemma 15). We will use standard matrix terminilogy: m_{ij} denotes the entry in the i-th row and j-th column of M, and $M_{i\cdot}$ is the i-th row of M.

▶ **Definition 14.** Let M be an $n \times m$ matrix of rank ρ. Let $z \in [1, \infty)$, and $\alpha, \beta \geq 1$. An $n \times \rho$ matrix A is an (α, β, z)-conditioned basis for M if the column vectors of A span the column space of M, and additionally A satisfies that: (1) $\sum_{i,j} |a_{ij}|^z \leq \alpha^z$, (2) for all $u \in \mathbb{R}^\rho$, $\| u \|_{z'} \leq \beta \| Au \|_z$, where $\| \cdot \|_{z'}$ is the dual norm for $\| \cdot \|_z$ (*i.e.* $1/z + 1/z' = 1$).

▶ **Lemma 15.** *Let M be an $n \times m$ matrix of rank ρ. Let $z \in [1, \infty)$. Let A be an (α, β, z)-conditioned basis for M. For every vector $u \in \mathbb{R}^m$, the following inequality holds: $|M_{i\cdot} u|^z \leq (\| A_{i\cdot} \|_z^z \cdot \beta^z) \| Mu \|_z^z$.*

Proof. We have $M = A\tau$ for some $\rho \times m$ matrix τ. Then,

$$|M_{i\cdot} u|^z = |A_{i\cdot} \tau u|^z \leq \| A_{i\cdot} \|_z^z \cdot \| \tau u \|_{z'}^z \leq \| A_{i\cdot} \|_z^z \cdot \beta^z \| A\tau u \|_z^z = \| A_{i\cdot} \|_z^z \cdot \beta^z \| Mu \|_z^z.$$

The second step is Holder's inequality, and the third uses the fact that A is (α, β, z)-conditioned. ◀

Using Lemma 15, we derive an upper bound on the total sensitivity when each shape is a hyperplane.

▶ **Lemma 16** (total sensitivity for fitting a hyperplane)**.** *Consider the shape fitting problem $(\mathbb{R}^d, \mathcal{F}, \text{dist})$ where $\mathcal{F}$ is the set of all $(d-1)$-flats, that is, hyperplanes. The total sensitivity of any n-point set is $O(d^{1+z/2})$ for $1 \leq z < 2$, $O(d)$ for $z = 2$, and $O(d^z)$ for $z > 2$.*

Proof. We can parameterize a hyperplane with a vector in $\mathbb{R}^{d+1}$, $u = \begin{bmatrix} u_1 & \cdots & u_{d+1} \end{bmatrix}^T$: the hyperplane determined by u is $h_u = \{x \in \mathbb{R}^d | \sum_{i=1}^d u_i x_i + u_{d+1} = 0\}$, where x_i denotes the i^{th} entry of the vector x. Without loss of generality, we may assume that $\sum_{i=1}^d u_i^2 = 1$. The Euclidean distance to h_u from a point $q \in \mathbb{R}^d$ is $\text{dist}(q, h_u) = |\sum_{i=1}^d u_i q_i + u_{d+1}| / \sqrt{\sum_{i=1}^d u_i^2} = |\sum_{i=1}^d u_i q_i + u_{d+1}|$. (the second equality follows from the assumption that $\sum_{i=1}^d u_i^2 = 1$.)

Let $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^d$ be any set of n points. Let $\tilde{p_i}$ denote the row vector $\begin{bmatrix} p_i^T & 1 \end{bmatrix}$, and let M be the $n \times (d+1)$ matrix whose i^{th} row is $\tilde{p_i}$. Then, $\text{dist}(p_i, h_u) = |M_{i\cdot} u|^z$, and $\text{dist}(P, h_u) = \sum_{i=1}^n |M_{i\cdot} u|^z = \| Mu \|_z^z$. Then using Lemma 15, we have $\sigma_P(p_i) = \sup_u \frac{|M_{i\cdot} u|^z}{\| Mu \|_z^z} \leq \| A_{i\cdot} \|_z^z \cdot \beta^z$, where A is an (α, β, z)-conditioned basis for M. Thus,

$$\mathfrak{S}_P = \sum_{i=1}^n \sigma_P(p_i) \leq \beta^z \sum_{i=1}^n \| A_{i\cdot} \|_z^z = \beta^z \sum_{i,j} |a_{ij}|^z = (\alpha\beta)^z.$$

For $1 \leq z < 2$, M has $((d+1)^{1/z+1/2}, 1, z)$-conditioned basis; for $z = 2$, M has $((d+1)^{1/2}, 1, z)$-conditioned basis; for $z > 2$, M has $((d+1)^{1/z+1/2}, (d+1)^{1/z'-1/2}, z)$-conditioned basis [5]. Thus the total sensitivity for the three cases are $(d+1)^{1+z/2}$, $d+1$, and $(d+1)^z$, respectively. ◀

It is now easy to derive dimension-dependent bounds on the sensitivity when each shape is a j-subspace.

▶ **Corollary 17** (Total sensitivity for fitting a j-subspace). *Consider the shape fitting problem* $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$ *where* $\mathcal{F}$ *is the set of all* j*-flats. The total sensitivity of any* n*-point set is* $O(d^{1+z/2})$ *for* $1 \leq z < 2$*,* $O(d)$ *for* $z = 2$*, and* $O(d^z)$ *for* $z > 2$*.*

Proof. Denote by $\mathcal{F}'$ the set of hyperplanes in $\mathbb{R}^d$. Let $P \subseteq \mathbb{R}^d$ be an arbitrary n-point set. We first show that $\sigma_{P,\mathcal{F}}(p) \leq \sigma_{P,\mathcal{F}'}(p)$, where the additional subscript is being used to indicate which shape fitting problem we are talking about (hyperplanes or j-flats). Let p be an arbitrary point in P. Let $F_p \in \mathcal{F}$ denote the j-subspace such that $\sigma_{P,\mathcal{F}}(p) = \mathrm{dist}(p, F_p)/\mathrm{dist}(P, F_p)$. Let $\mathrm{proj}\,(p, F_p)$ denote the projection of p on F_p. Consider the hyperplane F' containing F_p and orthogonal to the vector $p - \mathrm{proj}\,(p, F_p)$. We have $\mathrm{dist}(p, F') = \mathrm{dist}(p, F_p)$, whereas $\mathrm{dist}(q, F') \leq \mathrm{dist}(q, F_p)$ for each $q \in P$. Therefore, $\sigma_{P,\mathcal{F}'}(p) \geq \mathrm{dist}(p, F')/\mathrm{dist}(P, F') \geq \mathrm{dist}(p, F_p)/\mathrm{dist}(P, F_p) = \sigma_{P,\mathcal{F}}(p)$. It follows that $\mathfrak{S}_{P,\mathcal{F}} \leq \mathfrak{S}_{P,\mathcal{F}'}$. The statement in the corollary now follows from Lemma 16. ◀

6.2 Dimension-independent Bounds on the Sensitivity

We now derive dimension-independent upper bounds for the total sensitivity for the j-subspace fitting problem.

▶ **Theorem 18** (Total sensitivity for j-subspace fitting problem). *Consider the shape fitting problem* $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$ *where* $\mathcal{F}$ *is the set of all* j*-flats. The total sensitivity of any* n*-point set is* $O(j^{1+z/2})$ *for* $1 \leq z < 2$*,* $O(j)$ *for* $z = 2$*, and* $O(j^z)$ *for* $z > 2$*.*

Proof. Use Theorem 7, note that the projected point set P' is contained in a j-subspace. Further, each shape is a j-subspace. So, applying Theorem 8 and Corollary 17, the total sensitivity is $O(j^{2+z/2})$ or $z \in [1, 2)$, $O(j)$ for $z = 2$ and $O(j^z)$ for $z > 2$. ◀

Using Theorem 18 and the fact that $\dim(P)$ for the j-subspace fitting problem is $O(jd)$, we obtain small ϵ-coresets:

▶ **Theorem 19** (ϵ-coreset for j-subspace fitting problem). *Consider the shape fitting problem* $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$ *where* $\mathcal{F}$ *is the set of all* j*-flats. For any* n*-point set, there exists an* ϵ*-coreset whose size is* $O(j^{3+z}d\epsilon^{-2})$ *for* $z \in [1, 2)$*,* $O(j^3 d\epsilon^{-2})$ *for* $z = 2$ *and* $O(j^{2z+1}d\epsilon^{-2})$ *for* $z \geq 2$*.*

Proof. The result follows from Theorem 18, and Theorem 4. ◀

We note that for the case $j = d - 1$ and $z = 2$, a linear algebraic result from [2] yields a coreset whose size is an improved $O(d\epsilon^{-2})$.

7 The (j, k) integer projective clustering

▶ **Theorem 20.** *Consider the shape fitting problem* $(\mathbb{R}^d, \mathcal{F}, \mathrm{dist})$*, where* $\mathcal{F}$ *is the set of* k*-tuples of* j*-flats. Let* $P \subset \mathbb{R}^d$ *be any* n*-point instance with integer coordinates, the magnitude of each coordinate being at most* n^c*, for some constant* c*. The total sensitivity* $\mathfrak{S}_P$ *of* P *is* $O((\log n)^{f(k,j)})$*, where* $f(k, j)$ *is a function of only* k *and* j*. There exists an* ϵ*-coreset for* P *of size* $O((\log n)^{2f(k,j)}kjd\epsilon^{-2})$*.*

Proof. Observe that the projected point set $P' = \mathrm{proj}\,(P, \{J_1^*, \cdots, J_k^*\})$, where $\{J_1^*, \cdots, J_k^*\}$ is an optimum k-tuple of j-flats fitting P, is contained in a subspace of dimension $O(jk)$. Using Theorem 5, Theorem 8, and Theorem 7, the total sensitivity $\mathfrak{S}_P$ is upper bouned by

$O((\log n)^{f(k,j)})$, where $f(k,j)$ is a function of k and j. (A technical complication is that the coordinates of P', in the appropriate orthonormal basis, may not be integers. This can be addressed by rounding them to integers, at the expense of increasing the constant c. A similar procedure is adopted in [12], and we omit the details here.)

Using Theorem 4 and the fact that $\dim(P)$ is $O(djk)$, we obtain the bound on the coreset. ◀

8 Acknowledgements.

We thank the anonymous reviewers and Dan Feldman for their insightful feedback.

References

1 Pankaj K. Agarwal and Nabil H. Mustafa. k-Means projective clustering. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '04, pages 155–165, New York, NY, USA, 2004. ACM.

2 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. In *STOC*, pages 255–262, 2009.

3 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009.

4 Kenneth L. Clarkson. Subgradient and sampling algorithms for ℓ_1 regression. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 257–266, 2005.

5 Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W. Mahoney. Sampling algorithms and coresets for ℓ_p regression. *SIAM J. Comput.*, 38(5):2060–2078, 2009.

6 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *STOC*, pages 569–578. For an updated version, see `http://arxiv.org/abs/1106.1379v1`, 2011.

7 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, pages 11–18, New York, NY, USA, 2007. ACM.

8 Dan Feldman and Leonard J. Schulman. Data reduction for weighted and outlier-resistant clustering. In *SODA*, pages 1343–1354, 2012.

9 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, STOC '04, pages 291–300, New York, NY, USA, 2004. ACM.

10 Michael Langberg and Leonard J. Schulman. Universal ϵ-approximators for integrals. In *SODA*, pages 598–607, 2010.

11 Michael W. Mahoney and Petros Drineas. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

12 Kasturi Varadarajan and Xin Xiao. A near-linear algorithm for projective clustering integer points. In *SODA*, pages 1329–1342, 2012.

Overlap of Convex Polytopes under Rigid Motion*

Hee-Kap Ahn[1], Siu-Wing Cheng[2], Hyuk Jun Kweon[1], and Juyoung Yon[2]

1 **Department of Computer Science and Engineering, POSTECH, Korea.**
{heekap,kweon7182}@postech.ac.kr
2 **Department of Computer Science and Engineering, HKUST, Hong Kong.**
{scheng,jyon}@cse.ust.hk.

Abstract

We present an algorithm to compute an approximate overlap of two convex polytopes P_1 and P_2 in $\mathbb{R}^3$ under rigid motion. Given any $\varepsilon \in (0, 1/2]$, our algorithm runs in $O(\varepsilon^{-3} n \log^{3.5} n)$ time with probability $1 - n^{-O(1)}$ and returns a $(1-\varepsilon)$-approximate maximum overlap, provided that the maximum overlap is at least $\lambda \cdot \max\{|P_1|, |P_2|\}$ for some given constant $\lambda \in (0, 1]$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases convex polytope, overlap, approximation, rigid motion

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.498

1 Introduction

Shape matching is a common task in many object recognition applications. The particular problem of matching convex shapes has been used in tracking regions in an image sequence [8] and measuring symmetry of a convex body [6]. A translation or rigid motion of one shape is sought to maximize some similarity measure with another shape. The *overlap* of the two convex shapes—the volume of their intersection—is a robust similarity measure [12]. In this paper, we consider the problem of approximating the maximum overlap of two convex polytopes in $\mathbb{R}^3$ under rigid motion.

Efficient algorithms have been developed for two convex polygons of n vertices in the plane. De Berg et al. [5] developed an algorithm to find the maximum overlap of two convex polygons under translation in $O(n \log n)$ time. Ahn et al. [3] presented two algorithms to find a $(1-\varepsilon)$-approximate maximum overlap, one for the translation case and another for the rigid motion case. They assume that the polygon vertices are stored in arrays in clockwise order around the polygon boundaries. Ahn et al.'s algorithms run in $O(\varepsilon^{-1} \log n + \varepsilon^{-1} \log(1/\varepsilon))$ time for the translation case and $O(\varepsilon^{-1} \log n + \varepsilon^{-2} \log(1/\varepsilon))$ time for the rigid motion case. Finding the exact maximum overlap under rigid motion seems difficult. A brute force approach is to subdivide the space of rigid motion $[-\pi, \pi] \times \mathbb{R}^2$ into cells so that the intersecting pairs of polygon edges do not change within a cell. The hope is to obtain a formula for maximum overlap within a cell as the intersection does not change combinatorially. Unfortunately, the subdivision of $[-\pi, \pi] \times \mathbb{R}^2$ has curved edges and facets, which makes it a challenge to obtain formulae for maximum overlap in the cells.

Fewer algorithmic results are known concerning the maximum overlap of two convex polytopes in $\mathbb{R}^d$ for $d \geqslant 3$. Let n be the number of hyperplanes defining the convex polytopes.

* Research of Cheng and Yon was partly supported by the Research Grant Council, Hong Kong, China, project no. 611711. Research by Ahn was partly supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

Ahn et al. [4] developed an algorithm to find the maximum overlap of two convex polytopes under translation in $O(n^{(d^2+d-3)/2}\log^{d+1} n)$ expected time. Recently, Ahn, Cheng and Reinbacher [2] have obtained substantially faster algorithms to align two convex polytopes under translation in $\mathbb{R}^3$ and $\mathbb{R}^d$ for $d \geqslant 4$. In both cases, the overlap computed is no less than the optimum minus ε, where ε is an arbitrarily small constant fixed in advanced. The running times are $O(n\log^{3.5} n)$ for $\mathbb{R}^3$ and $O(n^{\lfloor d/2\rfloor+1}\log^d n)$ for $d \geqslant 4$, and these time bounds hold with probability $1-n^{-O(1)}$. There is no specific prior result concerning the maximum overlap of convex polytopes under rigid motion. Vigneron [13] studied the optimization of algebraic functions and one of the applications is the alignment of two possibly non-convex polytopes under rigid motion. For any $\varepsilon \in (0,1)$ and for any two convex polytopes with n defining hyperplanes, Vigneron's method can return in $O\big(\varepsilon^{-\Theta(d^2)}n^{\Theta(d^3)}(\log\frac{n}{\varepsilon})^{\Theta(d^2)}\big)$ time an overlap under rigid motion that is at least $1-\varepsilon$ times the optimum. Finding the exact overlap is even more challenging in $\mathbb{R}^3$.

In this paper, we present a new algorithm to approximate the maximum overlap of two convex polytopes P_1 and P_2 in $\mathbb{R}^3$ under rigid motion. For the purpose of shape matching, it often suffices to know that two input shapes are very dissimilar if this is the case. Therefore, we are only interested in matching P_1 and P_2 when their maximum overlap under rigid motion is at least $\lambda\cdot\max\{|P_1|, |P_2|\}$ for some given constant $\lambda \in (0,1]$, where $|P_i|$ denotes the volume of P_i. Under this assumption, given any $\varepsilon \in (0,1/2]$, our algorithm runs in $O(\varepsilon^{-3}n\log^{3.5} n)$ time with probability $1-n^{-O(1)}$ and returns a rigid motion that achieves a $(1-\varepsilon)$-approximate maximum overlap. The assumption can be verified as follows. Run our algorithm using $\lambda/2$ instead of λ. Check if the overlap output by our algorithm is at least $(1-\varepsilon)\lambda\cdot\max\{|P_1|, |P_2|\}$. If not, we know that the assumption is not satisfied. If yes, the maximum overlap is at least $(\lambda/2)\cdot\max\{|P_1|, |P_2|\}$ and our algorithm's output is a $(1-\varepsilon)$-approximation because we used $\lambda/2$ in running the algorithm.

Our high-level strategy has two steps. First, sample a set of rotations. Second, for each sampled rotation, apply it and then apply the almost optimal translation computed by Ahn et al.'s algorithm [2]. Finally, return the best answer among all rigid motions tried. If one uses a very fine uniform discretization of the rotation space, it is conceptually not difficult to sample rotations so that the resulting approximation is good. The problem is that such a discretization inevitably leads to a running time that depends on some geometric parameters of P_1 and P_2. In order to obtain a running time that depends on n and ε only, we cannot use a uniform discretization of the entire rotation space. Indeed, our contribution lies in establishing some structural properties that allow us to discretize a small subset of the rotation space, and exploiting this discretization in the analysis to prove the desired approximation. This approach is also taken in the 2D case in [3], but our analysis is not an extension of that in [3] as the three-dimensional situation is different and more complicated.

2 Similar Polytopes

In this section, we show that P_1 and P_2 are "similar" under the assumption that their maximum overlap is at least $\lambda\cdot\max\{|P_1|, |P_2|\}$. We use the Löwner-John ellipsoid [11] to identify the three axes of P_1 and P_2. For every convex body P in $\mathbb{R}^d$, it is proven by Löwner that there is a unique ellipsoid E containing P with minimum volume. Then John proved that $\frac{1}{d}E$ is contained in P. There are various algorithms for finding an ellipsoid of this flavor.

▶ **Lemma 1** ([11]). *Let P be a convex body with m vertices in $\mathbb{R}^3$. For every $\eta > 0$, an ellipsoid $\mathcal{E}(P)$ can be computed in $O(m/\eta)$ time such that $\frac{1}{3(1+\eta)}\mathcal{E}(P) \subset P \subset \mathcal{E}(P)$.*

For $i \in \{1, 2\}$, we use $\mathcal{E}(P_i)$ to denote the ellipsoid guaranteed by Lemma 1 for P_i. There are three mutually orthogonal directed lines α_i, β_i and γ_i through the center of $\mathcal{E}(P_i)$ such that $|\alpha_i \cap \mathcal{E}(P_i)|$ and $|\gamma_i \cap \mathcal{E}(P_i)|$ are the shortest and longest, respectively, among all possible directed lines through the center of $\mathcal{E}(P_i)$. After fixing α_i and γ_i, there are two choices for β_i and any one will do. We call these lines the α_i-, β_i-, and γ_i-axes of P_i. The lengths $a_i = |\alpha_i \cap \mathcal{E}(P_i)|$, $b_i = |\beta_i \cap \mathcal{E}(P_i)|$, and $c_i = |\gamma_i \cap \mathcal{E}(P_i)|$ are the three *principal diameters* of $\mathcal{E}(P_i)$. Notice that $a_i \leqslant b_i \leqslant c_i$. Define $a_{\min} = \min\{a_1, a_2\}$, $a_{\max} = \max\{a_1, a_2\}$, $b_{\min} = \min\{b_1, b_2\}$, $b_{\max} = \max\{b_1, b_2\}$, $c_{\min} = \min\{c_1, c_2\}$, and $c_{\max} = \max\{c_1, c_2\}$. The following technical result will be needed.

▶ **Lemma 2.** *For $i \in \{1, 2\}$, let R_i be a box with side lengths a_i, b_i, and c_i. The maximum overlap of R_1 and R_2 under rigid motion is at most $\sqrt{3}a_{\min}b_{\min}c_{\min}$.*

Proof. Without loss of generality, we suppose that a_1 is $a_{\min}$, that is, $a_1 \leq a_2$. If $b_{\min} = b_1$ and $c_{\min} = c_1$, then the maximum overlap of R_1 and R_2 are automatically $a_{\min}b_{\min}c_{\min}$ or less. There are three cases left: (i) $b_{\min} = b_2$ and $c_{\min} = c_2$, (ii) $b_{\min} = b_1$ and $c_{\min} = c_2$, and (iii) $b_{\min} = b_2$ and $c_{\min} = c_1$.

Let the ab-, bc-, and ca-planes of R_i be the plane through the center of R_i and parallel to the facets of side length a_i and b_i, b_i and c_i, and c_i and a_i respectively, where $i \in \{1, 2\}$. Place R_1 and R_2 such that their overlap is maximum.

Case 1: $b_{\min} = b_2$ and $c_{\min} = c_2$. Let θ be the nonobtuse angle between the normal lines of bc-plane of R_1 and the ab-plane of R_2.

Suppose that $\theta \leqslant \pi/4$. Refer to Figure 1(a). Consider the two facets of R_1 that are parallel to its bc-plane. The supporting planes of these two facets bound an infinite slab with width a_1. Consider the facets of R_2 that are parallel to its ab-plane. Sweeping these two facets along the normal line of the ab-plane of R_2 produces an infinite rectangular cylinder. The intersection of the slab and the cylinder is a parallelepiped that contains $R_1 \cap R_2$, We can assume that b_2 is parallel to the bc-plane of R_1 because the volume of the parallelepiped does not change while we rotate the cylinder around the normal line of ab-plane of R_2. Then, the parallelepiped's volume is $a_1a_2b_2/\cos\theta \leqslant a_1b_2c_2/\cos\theta \leqslant \sqrt{2}a_{\min}b_{\min}c_{\min}$.

Suppose that $\theta > \pi/4$. Refer to Figure 1(b). The angle between the normal lines of the bc-plane of R_1 and the bc-plane of R_2 is $\pi/2 - \theta$. We construct the infinite slab as in the above. We sweep the two facets of R_2 that are parallel to its bc-plane to obtain an infinite rectangular cylinder instead. The volume of the parallelepiped at the intersection of the slab and this new cylinder is $a_1b_2c_2/\cos(\pi/2 - \theta) \leqslant \sqrt{2}a_{\min}b_{\min}c_{\min}$.

Case 2: $b_{\min} = b_1$ and $c_{\min} = c_2$. Let θ be the nonobtuse angle between the normal lines of the ab-planes of R_1 and R_2.

Suppose that $\theta \leqslant \arccos(1/\sqrt{3})$. Refer to Figure 1(c). Consider the two facets of R_2 that are parallel to its ab-plane. The supporting planes of these two facets bound an infinite slab with width c_2. Consider the facets of R_1 that are parallel to its ab-plane. Sweeping these two facets along the normal line of the ab-plane of R_1 produces an infinite rectangular cylinder. As in case 1, we can assume that b_1 is parallel to the ab-plane of R_2. The intersection of the slab and the cylinder is a parallelepiped that contains $R_1 \cap R_2$, and the parallelepiped's volume is $a_1b_1c_2/\cos\theta \leqslant \sqrt{3}a_{\min}b_{\min}c_{\min}$.

Suppose that $\theta > \arccos(1/\sqrt{3})$. Refer to Figure 1(d). Let φ be the nonobtuse angle between the normal lines of the ab-plane of R_1 and the bc-plane of R_2. Let ψ be the nonobtuse angle between the normal lines of the ab-plane of R_1 and the ac-plane of R_2. Since $\cos^2\theta + \cos^2\varphi + \cos^2\psi = 1$, the sum $\cos^2\varphi + \cos^2\psi$ is at least $2/3$, which implies that $\cos\varphi$

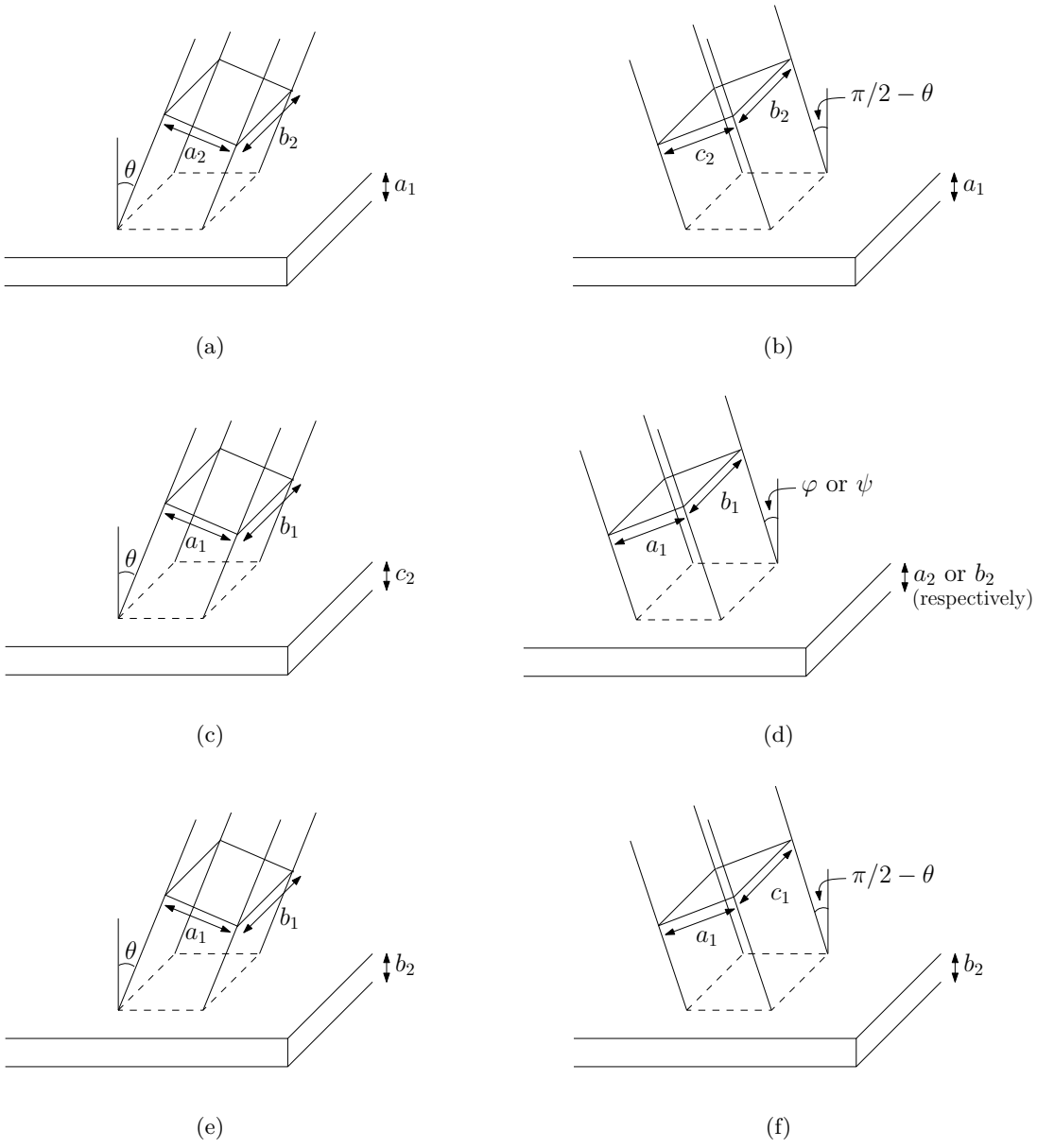


Figure 1 Illustrations for the proof of Lemma 2.

or $\cos\psi$ is at least $1/\sqrt{3}$. We construct the infinite rectangular cylinder as in the previous paragraph. If $\cos\varphi \geqslant 1/\sqrt{3}$, we take the slab bounded by the supporting planes of the two facets of R_2 that are parallel to its bc-plane. If $\cos\psi \geqslant 1/\sqrt{3}$, we take the slab bounded by the supporting planes of the two facets of R_2 that are parallel to its ac-plane. $R_1 \cap R_2$ is contained in the parallelepiped at the intersection of the slab and the cylinder, whose volume is at most $a_1a_2b_1/\cos\varphi$ if $\cos\varphi \geqslant 1/\sqrt{3}$, or $a_1b_1b_2/\cos\varphi$ if $\cos\varphi \geqslant 1/\sqrt{3}$. In either case, the volume is at most $\sqrt{3}a_1b_1c_2 = \sqrt{3}a_{\min}b_{\min}c_{\min}$.

Case 3: $b_{\min} = b_2$ and $c_{\min} = c_1$. Let θ be the nonobtuse angle between the normal lines of the ab-plane of R_1 and the ac-plane of R_2.

Suppose that $\theta \leqslant \pi/4$. Refer to Figure 1(e). Consider the two facets of R_2 that are parallel to its ac-plane. The supporting planes of these two facets bound an infinite slab with width b_2. Consider the facets of R_1 that are parallel to its ab-plane. Sweeping these two facets along the normal line of the ab-plane of R_1 produces an infinite rectangular cylinder. As in case 1, we can assume that a_1 is parallel to the ac-plane of R_2. The intersection of the slab and the cylinder is a parallelepiped that contains $R_1 \cap R_2$, and the parallelepiped's volume is $a_1b_1b_2/\cos\theta \leqslant a_1c_1b_2/\cos\theta \leqslant \sqrt{2}a_{\min}b_{\min}c_{\min}$.

Suppose that $\theta > \pi/4$. Refer to Figure 1(f). We keep the same slab in the previous paragraph. Sweep the two facets of R_1 that are parallel to its ac-plane to obtain an infinite rectangular cylinder. $R_1 \cap R_2$ is contained in the parallelepiped at the intersection of the slab and the cylinder. This parallelepiped has volume $a_1c_1b_2/\cos(\pi/2-\theta) \leqslant \sqrt{2}a_{\min}b_{\min}c_{\min}$. ◀

We are ready to show that P_1 and P_2 are similar in the sense that the respective principal diameters are within a constant factor of each other.

▶ **Lemma 3.** *If the maximum overlap of P_1 and P_2 under rigid motion is $\lambda \cdot \max\{|P_1|, |P_2|\}$ or more, then the ratios a_1/a_2, b_1/b_2, and c_1/c_2 are between $\lambda/(2^7\sqrt{3})$ and $2^7\sqrt{3}/\lambda$.*

Proof. It follows from Lemma 1 that $|P_i| \geq \frac{4}{3}\pi \cdot 2^{-3}3^{-3}(1+\eta)^{-3} \cdot a_ib_ic_i$ for $i \in \{1,2\}$. By setting η such that $3(1+\eta) < 4$, we obtain: for $i \in \{1,2\}$, $|P_i| \geq \frac{4}{3}\pi \cdot 2^{-3}4^{-3} \cdot a_ib_ic_i$. The maximum overlap of P_1 and P_2 under rigid motion is at most $\sqrt{3}a_{\min}b_{\min}c_{\min}$ by Lemma 2. Thus, for $i \in \{1,2\}$, $\sqrt{3}a_{\min}b_{\min}c_{\min} \geq \lambda|P_i| \geq (3/\pi) \cdot \lambda|P_i| \geqslant \lambda a_ib_ic_i/2^7$. It follows that

$$a_1/a_2 \leqslant (2^7\sqrt{3}/\lambda) \cdot (b_{\min}/b_1) \cdot (c_{\min}/c_1) \leqslant 2^7\sqrt{3}/\lambda \quad (1)$$

$$a_1/a_2 \geqslant (\lambda/(2^7\sqrt{3})) \cdot (b_1/b_{\min}) \cdot (c_1/c_{\min}) \geqslant \lambda/(2^7\sqrt{3}) \quad (2)$$

We can similarly show that b_1/b_2 and c_1/c_2 are between $\lambda/(2^7\sqrt{3})$ and $2^7\sqrt{3}/\lambda$. ◀

3 Sampling Rigid Motions

A rigid motion can be viewed as a rotation of P_1 and P_2 followed by a translation of P_2. A rotation is a relative motion between P_1 and P_2, and indeed, it is more convenient to rotate both P_1 and P_2 for our purposes. We set the initial positions of P_1 and P_2 so that the centers of $\mathcal{E}(P_1)$ and $\mathcal{E}(P_2)$ coincide and the α_1- and α_2-axes, β_1- and β_2-axes, and the γ_1- and γ_2-axes are aligned, respectively. Without loss of generality, assume that the $\alpha_1\beta_1$-plane is horizontal initially. (So is the $\alpha_2\beta_2$-plane as it coincides with the $\alpha_1\beta_1$-plane initially.)

A rotation R_* acts on the pair (P_1, P_2) and produces a new pair $R_*(P_1, P_2)$. R_* is decomposed into three simpler rotations R_β, R_α, and R_γ parametrized by three angles $\theta_\beta, \theta_\alpha, \theta_\gamma \in [-\pi, \pi]$, respectively, such that $R_\gamma(R_\beta(P_1))$ is the rotated P_1 by R_* and $R_\alpha(P_2)$ is the rotated P_2 by R_*. Let $\angle(u, v)$ be the angle between two oriented axes u and v. So

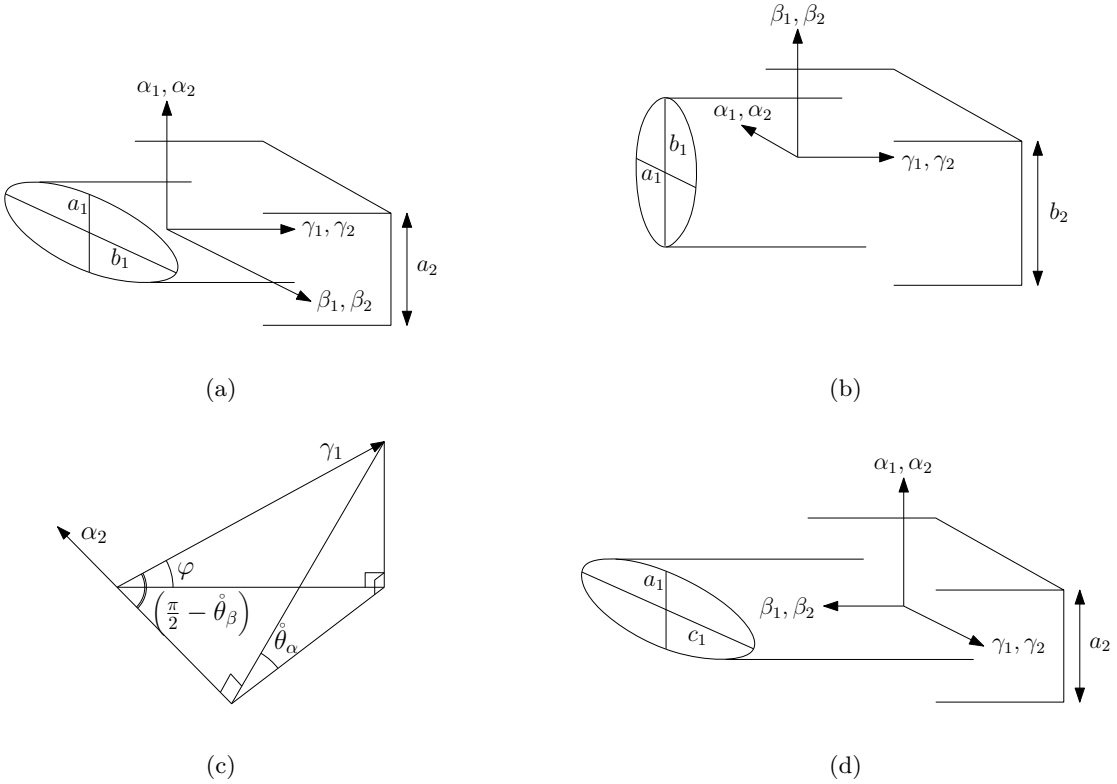


Figure 2 Illustrations for the proof of Lemma 4.

$\angle(u, v) \in [0, \pi]$. The idea is to first rotate P_1 and P_2 to fix the angle between γ_1 and the $\alpha_2\beta_2$-plane, and then rotate P_1 around γ_1. The detailed specification of R_* is as follows.

1. Rotate P_1 around the β_1-axis in the clockwise direction as viewed from infinity in β_1's direction by the angle θ_β. This is the rotation R_β which fixes the angle $\angle(\gamma_1, \alpha_2)$.
2. Rotate P_2 around the α_2-axis in the clockwise direction as viewed from infinity in α_2's direction by the angle θ_α. This is the rotation R_α which fixes the angle $\angle(\gamma_1, \beta_2)$. Notice that the angle $\angle(\gamma_1, \alpha_2)$ is not affected by R_α.
3. Rotate P_1 around the γ_1-axis in the clockwise direction as viewed from infinity in γ_1's direction by the angle θ_γ. This is the rotation R_γ. Notice that the angles $\angle(\gamma_1, \alpha_2)$ and $\angle(\gamma_1, \beta_2)$ are unaffected by R_γ.

The order of the applications of R_β, R_α and R_γ matters—the result of applying R_β, R_α and R_γ in this order can differ from the result of applying the same three rotations in another order. Every rotation in $\mathbb{R}^3$ is specified by a triple $(\theta_\beta, \theta_\alpha, \theta_\gamma) \in [-\pi, \pi] \times [-\pi, \pi] \times [-\pi, \pi]$.

▶ **Lemma 4.** *Let P_1 and P_2 be convex polytopes in $\mathbb{R}^3$. Let $\mathring{R}_*$ be the rotation part of an optimal rigid motion $\mathring{M}$ that maximizes the intersection volume of P_1 and P_2. Let $\mathring{\theta}_\beta$, $\mathring{\theta}_\alpha$ and $\mathring{\theta}_\gamma$ be the three angles in the representation of $\mathring{R}_*$. If $2^{13}3^5 a_{\min} \leqslant \lambda^2 c_{\min}/\sqrt{2}$, then*

$$|\sin\mathring{\theta}_\beta| \leq \frac{2^{13}3^5 a_{\min}}{\lambda^2 c_{\min}}, \quad |\sin\mathring{\theta}_\alpha| \leq \frac{2^{14}3^5 b_{\min}}{\sqrt{2}\lambda^2 c_{\min}}, \quad |\sin\mathring{\theta}_\gamma| \leq \frac{2^{13}3^5 a_{\min}}{\lambda^2 b_{\min}}.$$

Proof. By Lemma 1, if we position P_1 and P_2 such that the centers of $\mathcal{E}(P_1)$ and $\mathcal{E}(P_2)$ coincide and the respective axes of P_1 and P_2 are aligned, the overlap of P_1 and P_2 contains an

ellipsoid with principal diameters $a_{\min}/3$, $b_{\min}/3$, and $c_{\min}/3$. Let $\mathring{T}$ be the translation part of the optimal rigid motion. Thus, $|\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{T}(\mathring{R}_\alpha(P_2))| \geq \frac{4}{3}\pi a_{\min}b_{\min}c_{\min}/(2^3 3^3)$.

Enclose $\mathcal{E}(P_1)$ in an elliptic cylinder C such that the base of C has principal diameters a_1 and b_1, and the axis of C is aligned with the γ_1-axis of P_1. Enclose $\mathcal{E}(P_2)$ with a infinite slab S that has thickness a_2 and is parallel to the $\beta_2\gamma_2$-plane. Refer to Figure 2(a). When we apply $\mathring{R}_*$, we get $|\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(P_2)| \leqslant |\mathring{R}_\gamma(\mathring{R}_\beta(C)) \cap \mathring{R}_\alpha(S)|$. Only $\mathring{R}_\beta$ has an effect on the volume of $|\mathring{R}_\gamma(\mathring{R}_\beta(C)) \cap \mathring{R}_\alpha(S)|$ because $\mathring{R}_\alpha$ does not change the shape of the intersection, and $\mathring{R}_\gamma$ does not change the volume of the intersection as long as $\mathring{\theta}_\beta \notin \{0, \pi, -\pi\}$. $\mathring{R}_\gamma(\mathring{R}_\beta(C)) \cap \mathring{R}_\alpha(S)$ has base area $\pi a_1 b_1/(2^2|\sin\mathring{\theta}_\beta|)$ and height a_2, so $|\mathring{R}_\gamma(\mathring{R}_\beta(C)) \cap \mathring{R}_\alpha(S)| = \pi a_1 a_2 b_1/(2^2|\sin\mathring{\theta}_\beta|)$. Obviously, applying the translation part $\mathring{T}$ of the optimal rigid motion to $\mathring{R}_\alpha(S)$ has no impact on the intersection volume. Therefore, $|\mathring{R}_\gamma(\mathring{R}_\beta(C)) \cap \mathring{R}_\alpha(S)| \geqslant |\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{T}(\mathring{R}_\alpha(P_2))| \geqslant \frac{4}{3}\pi a_{\min}b_{\min}c_{\min}/(2^3 3^3)$. We conclude that

$$\frac{4\pi}{2^3 3^4}\, a_{\min}b_{\min}c_{\min} \leqslant \frac{\pi}{2^2|\sin\mathring{\theta}_\beta|}\, a_1 a_2 b_1 \quad \Rightarrow \quad |\sin\mathring{\theta}_\beta| \leqslant \frac{3^4 a_1 a_2 b_1}{2a_{\min}b_{\min}c_{\min}} \leqslant \frac{2^{13}3^5 a_{\min}}{\lambda^2 c_{\min}}.$$

The last inequality follows from Lemma 3.

The assumption of $2^{13}3^5 a_{\min} \leqslant \lambda^2 c_{\min}/\sqrt{2}$ is needed in bounding $|\sin\mathring{\theta}_\alpha|$ and $|\sin\mathring{\theta}_\gamma|$. By this assumption, $|\sin\mathring{\theta}_\beta| \leqslant 1/\sqrt{2}$ and $\mathring{\theta}_\beta \in [0, \pi/4] \cup (-\pi, -3\pi/4]$. To bound $\sin\mathring{\theta}_\alpha$, we similarly enclose $\mathcal{E}(P_1)$ with an elliptic cylinder C and $\mathcal{E}(P_2)$ with a slab S, except that we swap the positions of α_i and β_i. The height of the slab S enclosing $\mathcal{E}(P_2)$ is thus b_2. Refer to Figure 2(b). R_α makes C tilt at an acute angle φ to S, while R_γ has no effect on the volume of the intersection of C and S as long as $\mathring{\theta}_\alpha \notin \{0, \pi, -\pi\}$. The maximum value of $\sin\varphi$ is $|\sin\mathring{\theta}_\alpha|$ when $\mathring{\theta}_\beta = 0$ or π or $-\pi$; and $\sin\varphi$ is minimized when $\mathring{\theta}_\beta = \pi/4$ or $-3\pi/4$. Refer to Figure 2(c); by elementary trigonometry, the minimum value of $\sin\varphi$ is $|\sin\mathring{\theta}_\alpha|/\sqrt{2}$. As in the last paragraph, $|\mathring{R}_\gamma(\mathring{R}_\beta(C)) \cap \mathring{R}_\alpha(S)| = \pi a_1 b_1 b_2/(2^2\sin\varphi) \leqslant \pi a_1 b_1 b_2/(2\sqrt{2}|\sin\mathring{\theta}_\alpha|)$. Therefore,

$$\frac{4\pi}{2^3 3^4}\, a_{\min}b_{\min}c_{\min} \leqslant \frac{\pi}{2\sqrt{2}|\sin\mathring{\theta}_\alpha|}\, a_1 b_1 b_2 \quad \Rightarrow \quad |\sin\mathring{\theta}_\alpha| \leqslant \frac{3^4 a_1 b_1 b_2}{\sqrt{2}a_{\min}b_{\min}c_{\min}} \leqslant \frac{2^{14}3^5 b_{\min}}{\sqrt{2}\lambda^2 c_{\min}}.$$

The analysis for $\sin\mathring{\theta}_\gamma$ is similar. We enclose $\mathcal{E}(P_1)$ with an elliptic cylinder C and $\mathcal{E}(P_2)$ with a slab S as shown in Figure 2(d). Notice that the height of S is a_2. R_β has no effect on the volume of the intersection of C and S as long as $\mathring{\theta}_\gamma \notin \{0, \pi, -\pi\}$, and R_α does not change the shape of intersection of C and S. R_γ makes C tilt at an acute angle $\mathring{\theta}_\gamma$ to S. Therefore, $|\mathring{R}_\gamma(\mathring{R}_\beta(C)) \cap \mathring{R}_\alpha(S)| = \pi a_1 a_2 c_1/(2^2|\sin\mathring{\theta}_\gamma|)$. Therefore,

$$\frac{4\pi}{2^3 3^4}\, a_{\min}b_{\min}c_{\min} \leqslant \frac{\pi}{2^2|\sin\mathring{\theta}_\gamma|}\, a_1 a_2 c_1 \quad \Rightarrow \quad |\sin\mathring{\theta}_\gamma| \leqslant \frac{3^4 a_1 a_2 c_1}{2a_{\min}b_{\min}c_{\min}} \leqslant \frac{2^{13}3^5 a_{\min}}{\lambda^2 b_{\min}}.$$

◀

Lemma 4 tells us that if $2^{13}3^5 a_{\min} \leqslant \lambda^2 c_{\min}/\sqrt{2}$, the angles θ_β, θ_α and θ_γ can only vary in some appropriate subsets of $[-\pi, \pi]$. This allows us to discretize only a small subrange of $[-\pi, \pi]$ in designing our approximation algorithm, which helps to reduce the running time. If $2^{13}3^5 a_{\min} > \lambda^2 c_{\min}/\sqrt{2}$, the lengths a_i, b_i and c_i are within constant factors of each other, and it suffices to discretize the range $[-\pi, \pi]$ uniformly in this case. In the following, we first define the angular ranges I_β, I_α and I_γ for θ_β, θ_α and θ_γ respectively, and then discuss the discretization of these three ranges.

- If $2^{13}3^5 a_{\min} > \lambda^2 c_{\min}/\sqrt{2}$, then $I_\beta = I_\alpha = I_\gamma = [-\pi, \pi]$.

- If $2^{13}3^5 a_{\min} \leqslant \lambda^2 c_{\min}/\sqrt{2}$, then for $\xi \in \{\beta, \alpha, \gamma\}$,

$$I_\xi = [0, f_\xi] \cup [\pi - f_\xi, \pi] \cup [-f_\xi, 0] \cup [-\pi, -\pi + f_\xi],$$

where:

- $f_\beta = \arcsin(2^{13}3^5 a_{\min}/(\lambda^2 c_{\min}))$.
- $f_\alpha = \arcsin(2^{14}3^5 b_{\min}/(\sqrt{2}\lambda^2 c_{\min}))$ if $2^{14}3^5 b_{\min} \leqslant \sqrt{2}\lambda^2 c_{\min}$; otherwise, $f_\alpha = \pi$.
- $f_\gamma = \arcsin(2^{13}3^5 a_{\min}/(\lambda^2 b_{\min}))$ if $2^{13}3^5 a_{\min} \leqslant \lambda^2 b_{\min}$; otherwise, $f_\gamma = \pi$.

The rotation part $\mathring{R}_*$ of the optimal rigid motion belongs to $I_\beta \times I_\alpha \times I_\gamma$ according to Lemma 4. We sample angle triples from $I_\beta \times I_\alpha \times I_\gamma$ at intervals of $\Delta_\beta\varepsilon, \Delta_\alpha\varepsilon, \Delta_\gamma\varepsilon$ respectively:

$$\Delta_\beta = \frac{a_{\min} b_{\min} c_{\min}}{2^4 3^5 b_1 c_1^2}, \quad \Delta_\alpha = \frac{1}{2} \cdot \frac{a_{\min} b_{\min} c_{\min}}{3^5 a_2 c_2^2}, \quad \Delta_\gamma = \frac{1}{2} \cdot \frac{a_{\min} b_{\min} c_{\min}}{3^5 b_1^2 c_1}.$$

Let S_ξ denote the set of angles sampled from I_ξ for $\xi \in \{\beta, \alpha, \gamma\}$. Our strategy is to try all rotations in $S_\beta \times S_\alpha \times S_\gamma$ and for each such rotation, find the best translation to maximize the overlap. It remains to show that the best rigid motion obtained by this strategy gives a $(1-\varepsilon)$-approximation.

In the lemma below, the rotation center p can be outside of C unlike in Lemma 4 from [1]. The proof is similar to that of Lemma 4 in [1].

▶ **Lemma 5.** *Let C be a convex set in $\mathbb{R}^2$, and let C' be a copy of C, rotated by an angle δ around a point p that is at distance l or less from any point in C. Then $|C \backslash C'| \leq (\pi\delta l/2) \cdot \mathrm{diam}(C) + \pi\delta^2 l^2/8$.*

Proof. We denote by D the symmetric difference between C and C'. Note that $|D| = |C| - |C \cap C'| + |C'| - |C \cap C'| = 2(|C| - |C \cap C'|) \geq 2|C \backslash C'|$. Let C_r be the rotated copy of C by an angle $\delta/2$ around p. Let T_r be the set of points that are at distance at most $\delta l/2$ from the boundary of C_r. Note that any point q in D is obtained from a point on the boundary of C_r by a rotation around p by an angle at most $\delta/2$. Since the distance $d(p, q)$ is at most l, q is an element of T_r. Thus $D \subset T_r$. Because the Minkowski sum of the boundary of C and a disk of radius r has area less than or equal to $2r\mathrm{peri}(C) + \pi r^2$, the area of T_r is at most $\delta l \mathrm{peri}(C) + \pi\delta^2 l^2/4$. Since $\mathrm{peri}(C_r) = \mathrm{peri}(C)$ and $\mathrm{peri}(C) \leq \pi \mathrm{diam}(C)$, we obtain that $|T_r| \leq \pi\delta l \mathrm{diam}(C) + \pi\delta^2 l^2/4$. Since $D \subset T_r$, it implies that $|D| \leq \pi\delta l \mathrm{diam}(C) + \pi\delta^2 l^2/4$. Therefore $|C \backslash C'| \leq \frac{1}{2}|D| \leq (\pi\delta l/2)\mathrm{diam}(C) + \pi\delta^2 l^2/8$. ◀

The following lemma is another extension of Lemma 4 in [1]. It shows that two copies of a convex polyhedron have small symmetric difference if there is a bound on the Hausdorff distance between them.

▶ **Lemma 6.** *Let C be a convex polyhedron in $\mathbb{R}^3$, and let C' be a copy of C such that the Hausdorff distance between C and C' is at most l. Let c and b be the first and second largest principal diameters of $\mathcal{E}(\mathcal{C})$. Then $|C \backslash C'| \leq \frac{4}{3}\pi l^3 + 2\pi bcl + 2\pi^2 cl^2$.*

Proof. Note that any point q in $C \backslash C'$ is in distance at most l from a point on the boundary of C'. Therefore $|C \backslash C'|$ has volume less than or equal to the volume of the Minkowski sum of the boundary of C' and a ball of radius l. Let V be the set of vertices, E be the set of edges, and F be the set of facets of C'. For every $j \in [0, 3]$ and every j-face f of C', define the interior angle $\varphi(f)$ to be the fraction of an arbitrarily small sphere centered at an interior point of f, that lies inside C'. For example, $\varphi(f) = 1$ if $f = C'$, $\varphi(f) = 1/2$ if f is a facet,

$\varphi(f)$ is the ratio of the internal directed angle at f to 2π if f is an edge, and $\varphi(f)$ is the solid angle at f if f is a vertex. The exterior angle $\theta(f)$ is defined to be $1/2 - \varphi(f)$. The Minkowski sum of the boundary of C' and a ball of radius l has volume less than

$$\frac{4}{3}\pi l^3 + 2l \cdot \sum_{f \in F} \text{area}(f) + \sum_{e \in E} \pi l^2 \cdot \text{length}(e) \cdot 2\pi\theta(e)$$

where area(f) is the area of a facet f and length(e) is the length of an edge e. Since C' is a convex polyhedron, the total surface area of C' does not exceed the surface area of the ellipsoid $E(C')$ containing C' which surface area is less than πbc [7] because $E(C')$ and $E(C)$ are identical. So we obtain the volume bound as below.

$$\frac{4}{3}\pi l^3 + 2\pi bcl + \sum_{e \in E} \pi l^2 \cdot \text{length}(e) \cdot 2\pi\theta(e)$$

The last term can be bounded as follow. By the Gram-Euler theorem [9, 10], we know that

$$\sum_{j \in [0,3]} \sum_{j\text{-face } f \text{ of } C'} (-1)^j \varphi(f) = 0.$$

Therefore,

$$\sum_{v \in V} \varphi(v) - \sum_{e \in E} \varphi(e) + \sum_{f \in F} 1/2 = 1.$$

Since $\sum_{v \in V} \theta(v) = 1$, $\sum_{v \in V} \varphi(v) = |V|/2 - \sum_{v \in V} \theta(v) = |V|/2 - 1$, and $\sum_{e \in E} \theta(e) = |E|/2 - \sum_{e \in E} \varphi(E) = 2 - (|V| - |E| + |F|)/2$. By the Euler's formula, $|V| - |E| + |F|$ is 2. It follows that $\sum_{e \in E} \theta(e) = 1$. Therefore, $|C \backslash C'| \leq \frac{4}{3}\pi l^3 + 2\pi bcl + \sum_{e \in E} \pi l^2 \cdot \text{length}(e) \cdot 2\pi\theta(e) \leq \frac{4}{3}\pi l^3 + 2\pi bcl + \sum_{e \in E} \pi cl^2 \cdot 2\pi\theta(e) = \frac{4}{3}\pi l^3 + 2\pi bcl + 2\pi^2 cl^2 \sum_{e \in E} \theta(e) = \frac{4}{3}\pi l^3 + 2\pi bcl + 2\pi^2 cl^2$. ◀

The next result proves the correctness of our strategy to find a $(1-\varepsilon)$-approximately maximum overlap of P_1 and P_2 under rigid motion.

▶ **Lemma 7.** *Let P_1 and P_2 be convex polytopes in $\mathbb{R}^3$. Let ε be a constant from the range $(0, 1/2)$. Suppose that the maximum overlap of P_1 and P_2 under rigid motion is at least $\lambda \cdot \max\{|P_1|, |P_2|\}$ for some constant $\lambda \in (0, 1]$. Then, there exists a rotation $\tilde{R}_* \in S_\beta \times S_\alpha \times S_\gamma$ and a translation $\tilde{T}$ such that $|\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{T}(\tilde{R}_\alpha(P_2))|$ is at least $1-\varepsilon$ times the maximum overlap of P_1 and P_2 under rigid motion.*

Proof. The rotation part $\mathring{R}_*$ of the optimal rigid motion is represented by a triple of angles $(\mathring{\theta}_\beta, \mathring{\theta}_\alpha, \mathring{\theta}_\gamma) \in I_\beta \times I_\alpha \times I_\gamma$. For $\xi \in \{\beta, \alpha, \gamma\}$, let $\tilde{\theta}_\xi$ be the closest interval endpoint in S_ξ to $\mathring{\theta}_\xi$. Then, $(\tilde{\theta}_\beta, \tilde{\theta}_\alpha, \tilde{\theta}_\gamma)$ defines a rotation $\tilde{R}_*$. Let $\tilde{R}_\alpha$, $\tilde{R}_\beta$, and $\tilde{R}_\gamma$ denote the three simple rotations that comprise $\tilde{R}_*$.

Let $\tilde{T}$ denote the translation that maximizes the overlap of $\tilde{R}_\gamma(\tilde{R}_\beta(P_1))$ and $\tilde{R}_\alpha(P_2)$. Let $\mathring{T}$ denote the translation that maximizes the overlap of $\mathring{R}_\gamma(\mathring{R}_\beta(P_1))$ and $\mathring{R}_\alpha(P_2)$. Therefore, $|\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{T}(\tilde{R}_\alpha(P_2))| \geq |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \mathring{T}(\tilde{R}_\alpha(P_2))|$. We analyze the difference between the maximum overlap and the approximate overlap as follows.

$$\begin{aligned}
& |\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{T}(\mathring{R}_\alpha(P_2))| - |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{T}(\tilde{R}_\alpha(P_2))| \\
\leqslant\ & |\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{T}(\mathring{R}_\alpha(P_2))| - |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \mathring{T}(\tilde{R}_\alpha(P_2))| \\
=\ & |\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))| - |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{R}_\alpha(\mathring{T}(P_2))| \\
=\ & |\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))| - |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))| + \quad (3) \\
& |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))| - |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))| + \quad (4) \\
& |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{R}_\alpha(\mathring{T}(P_2))| - |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{R}_\alpha(\mathring{T}(P_2))|. \quad (5)
\end{aligned}$$

If a point p lies in $\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))$ but not in $\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))$, then $p \in \mathring{R}_\gamma(\mathring{R}_\beta(P_1))$ but $p \notin \tilde{R}_\gamma(\mathring{R}_\beta(P_1))$. The common rotation $\mathring{R}_\beta$ can be ignored. Thus,

$$|\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))| \; - \; |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))| \; \leqslant \; |\mathring{R}_\gamma(P_1) \setminus \tilde{R}_\gamma(P_1)|.$$

Similar reasoning shows that

$$\begin{aligned} |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{R}_\alpha(\mathring{T}(P_2))| - |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \tilde{R}_\alpha(\mathring{T}(P_2))| \quad &\leqslant \quad |\mathring{R}_\alpha(P_2) \setminus \tilde{R}_\alpha(P_2)|, \\ |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \tilde{R}_\alpha(\mathring{T}(P_2))| - |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{R}_\alpha(\mathring{T}(P_2))| \quad &\leqslant \quad |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \setminus \tilde{R}_\gamma(\tilde{R}_\beta(P_1))|. \end{aligned}$$

Let H be a plane perpendicular to the γ_1-axis of P_1 that intersects $\mathring{R}_\gamma(P_1)$ and $\tilde{R}_\gamma(P_1)$. The convex polygon $H \cap \mathring{R}_\gamma(P_1)$ is rotated from the convex polygon $H \cap \tilde{R}_\gamma(P_1)$ by an angle at most $\varepsilon\Delta_\gamma$ around a point in $H \cap \tilde{R}_\gamma(P_1)$. The diameter of $H \cap \tilde{R}_\gamma(P_1)$ is at most b_1. Since the rotation center in $H \cap \tilde{R}_\gamma(P_1)$ is at distance at most $b_1/2$ from any point in $H \cap \mathring{R}_\gamma(P_1)$, Lemma 5 can be applied. Thus $|(H \cap \mathring{R}_\gamma(P_1)) \setminus (H \cap \tilde{R}_\gamma(P_1))| \leqslant (\pi/2)\varepsilon b_1^2\Delta_\gamma$, which implies that

$$|\mathring{R}_\gamma(P_1) \setminus \tilde{R}_\gamma(P_1)| \leqslant \int_{-c_1}^{c_1} \left(\frac{\pi}{2}\right) \cdot \varepsilon b_1^2\Delta_\gamma \; \mathrm{d}x = \pi\varepsilon b_1^2 c_1 \Delta_\gamma \leqslant \left(\frac{1}{3}\right) \cdot \frac{\pi}{2 \cdot 3^4} \varepsilon a_{\min} b_{\min} c_{\min}.$$

Similar reasoning shows that

$$|\mathring{R}_\alpha(P_2) \setminus \tilde{R}_\alpha(P_2)| \leqslant \int_{-a_2}^{a_2} \left(\frac{\pi}{2}\right) \cdot \varepsilon c_2^2\Delta_\alpha \; \mathrm{d}x = \pi\varepsilon a_2 c_2^2 \Delta_\alpha \leqslant \left(\frac{1}{3}\right) \cdot \frac{\pi}{2 \cdot 3^4} \varepsilon a_{\min} b_{\min} c_{\min}.$$

Substitute these results into (3)–(5) gives

$$\begin{aligned} & |\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{T}(\mathring{R}_\alpha(P_2))| \; - \; |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{T}(\tilde{R}_\alpha(P_2))| \\ \leqslant \; & |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \setminus \tilde{R}_\gamma(\tilde{R}_\beta(P_1))| \; + \; \left(\frac{2}{3}\right) \cdot \frac{\pi}{2 \cdot 3^4} \varepsilon a_{\min} b_{\min} c_{\min}. \end{aligned}$$

We bound $|\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \setminus \tilde{R}_\gamma(\tilde{R}_\beta(P_1))|$ as follows. This set difference may be non-empty because the γ_1-axis of $\mathring{R}_\beta(P_1)$ can make an angle up to $\varepsilon\Delta_\beta$ with the γ_1-axis of $\tilde{R}_\beta(P_1)$. This slight misalignment causes the results to be different after rotating $\mathring{R}_\beta(P_1)$ and $\tilde{R}_\beta(P_1)$ around their respective γ_1-axes by the same angle. Let x be a point of P_1. To apply the Lemma 6, we want to bound the distance between $\tilde{R}_\gamma(\mathring{R}_\beta(x))$ and $\tilde{R}_\gamma(\tilde{R}_\beta(x))$.

Let $\mathring{x}$ be $\mathring{R}_\beta(x)$ and let $\tilde{x}$ be $\tilde{R}_\beta(x)$. Then $\|\mathring{x} - \tilde{x}\| \leqslant \varepsilon c_1\Delta_\beta/2$. When $\tilde{R}_\gamma$ is applied, the point $\mathring{x}$ and $\tilde{x}$ are rotated in the planes orthogonal to the γ-axes of $\mathring{R}_\beta(P_1)$ and $\tilde{R}_\beta(P_1)$ respectively. We denote these planes $\mathring{H}$ and $\tilde{H}$, which contain $\mathring{x}$ and $\tilde{x}$ respectively and are orthogonal to the γ-axes of $\mathring{R}_\beta(P_1)$ and $\tilde{R}_\beta(P_1)$ respectively. Let $\mathring{c}$ be the intersection between $\mathring{H}$ and the γ-axis of $\mathring{R}_\beta(P_1)$. Let $\tilde{c}$ be the intersection of $\tilde{H}$ and the γ-axis of $\tilde{R}_\beta(P_1)$. Note that $\|\mathring{c} - \tilde{c}\| \leqslant \varepsilon c_1\Delta_\beta/2$, $\|\mathring{c} - \mathring{x}\| \leqslant b_1/2$, and $\|\tilde{c} - \tilde{x}\| \leqslant b_1/2$. Another fact is that $\|\mathring{c} - \mathring{x}\| = \|\tilde{c} - \tilde{x}\|$. Therefore, we can imagine that $\tilde{R}_\gamma$ rotates $\mathring{x}$ on the boundary of a disk $\mathring{D}$ on $\mathring{H}$ with center $\mathring{c}$ and radius $r \leqslant b_1/2$. Similarly, $\tilde{R}_\gamma$ rotates $\tilde{x}$ on the boundary of a disk $\tilde{D}$ on $\tilde{H}$ with center $\tilde{c}$ and radius r. The distance $\|\mathring{x} - \tilde{x}\|$ is at most $\varepsilon c_1\Delta_\beta/2$. Move $\mathring{D}$ to align the points $\mathring{c}$ and $\tilde{c}$ and also the points $\mathring{x}$ and $\tilde{x}$. Let D denote the moved $\mathring{D}$. Let y be the point on the boundary of D that correspoinds to $\tilde{R}_\gamma(\mathring{x})$. Let z be the point $\tilde{R}_\gamma(\tilde{x})$. By the triangle inequality, the distance between $\tilde{R}_\gamma(\mathring{R}_\beta(x))$ and $\tilde{R}_\gamma(\tilde{R}_\beta(x))$ is at most $\|\mathring{c} - \tilde{c}\| + \|\mathring{x} - \tilde{x}\| + \|y - z\| \leqslant \varepsilon c_1\Delta_\beta + \|y - z\|$. Using the spherical sine law, one can show that $\|y - z\| \leqslant (\pi b_1/2)\varepsilon\Delta_\beta$ because $\tilde{R}_\gamma$ rotates by an angle of magnitude π or less. So the distance been $\tilde{R}_\gamma(\mathring{R}_\beta(x))$ and $\tilde{R}_\gamma(\tilde{R}_\beta(x))$ is at most $((\pi/2) \cdot b_1 + c_1)\varepsilon\Delta_\beta < (2b_1 + c_1)\varepsilon\Delta_\beta$. The above relation holds for every point $x \in P_1$, which means that the Hausdorff distance between $\tilde{R}_\gamma(\mathring{R}_\beta(P_1))$ and $\tilde{R}_\gamma(\tilde{R}_\beta(P_1))$ is at most

$l \leqslant (2b_1+c_1)\varepsilon\Delta_\beta$. We apply the Lemma 6 with $C = \tilde{R}_\gamma(\mathring{R}_\beta(P_1))$ and $C' = \tilde{R}_\gamma(\tilde{R}_\beta(P_1))$. Note that $(2b_1 + c)\varepsilon\Delta_\beta \leqslant 3\varepsilon a_{min}b_{min}c_{min}/(2^4 3^5 b_1 c_1) \leqslant 3\varepsilon a_{min}c_{min}/(2^4 3^5 c_1) \leqslant 3\varepsilon a_{min}/(2^4 3^5)$. Lemma 6 gives

$$\begin{aligned} |\tilde{R}_\gamma(\mathring{R}_\beta(P_1)) \setminus \tilde{R}_\gamma(\tilde{R}_\beta(P_1))| &\leqslant \frac{4}{3}\pi l^3 + 2\pi b_1 c_1 l + 2\pi^2 c_1 l^2 \\ &< \varepsilon\pi(0.1\varepsilon^2 a_{min}^3 + 3a_{min}b_{min}c_{min} + 0.1\varepsilon a_{min}^2 c_{min})/(2^3 3^5) \\ &< \varepsilon\pi a_{\min}b_{\min}c_{\min}/(2\cdot 3^5). \end{aligned}$$

Hence, $|\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{T}(\mathring{R}_\alpha(P_2))| - |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{T}(\tilde{R}_\alpha(P_2))| < \varepsilon\pi a_{\min}b_{\min}c_{\min}/(2\cdot 3^4)$. Notice that $\frac{1}{3}\mathcal{E}(P_1) \cap \frac{1}{3}\mathcal{E}(P_2)$ lies inside $P_1 \cap P_2$ and has volume $\pi a_{\min}b_{\min}c_{\min}/(2\cdot 3^4)$. It follows that $|\mathring{R}_\gamma(\mathring{R}_\beta(P_1)) \cap \mathring{T}(\mathring{R}_\alpha(P_2))| - |\tilde{R}_\gamma(\tilde{R}_\beta(P_1)) \cap \tilde{T}(\tilde{R}_\alpha(P_2))|$ is at most ε times the maximum overlap of P_1 and P_2 under rigid motion. ◀

4 Main Algorithm

We use last section's result to sample a set of rotations $S_\beta \times S_\alpha \times S_\gamma$ from $I_\beta \times I_\alpha \times I_\gamma$. For each rotation $R_* \in S_\beta \times S_\alpha \times S_\gamma$, we want to compute the best translation to align $R_\gamma(R_\beta(P_1))$ and $R_\alpha(P_2)$, and then keep track of the rigid motion $M = (T, R_*)$ encountered so far that gives the largest overlap. For efficiency purpose, we compute the "almost best" translation using Theorem 8 below. Algorithm 1 shows the pseudocode of our algorithm.

▶ **Theorem 8** ([2]). *Let P_1 and P_2 be two convex polytopes in $\mathbb{R}^3$ specified by n bounding planes. For any $\mu > 0$, we can compute an overlap of P_1 and P_2 under translation that is at most μ less than the optimum. The running time is $O(n \log^{3.5} n)$ with probability $1 - n^{-O(1)}$.*

Algorithm 1 Maximum overlap approximation algorithm

```
1:  procedure MaxOverlap(P1, P2, ε)          ▷ return (1 − ε)-optimal rigid motion
2:      Compute E(P1) and E(P2) and align their centers and the respective axes.
3:      Compute three sets of sampled angles Sβ, Sα, and Sγ.
4:      ans := 0
5:      M := null
6:      for all rotation R* ∈ Sβ × Sα × Sγ do
7:          Compute the translation T to align Rγ(Rβ(P1)) and Rα(P2) using Theorem 8
8:          if |Rγ(Rβ(P1)) ∩ T(Rα(P2))| > ans then
9:              ans := |Rγ(Rβ(P1)) ∩ T(Rα(P2))|
10:             M := (R*, T)
11:         end if
12:     end for
13:     return M
14: end procedure
```

▶ **Theorem 9.** *Let P_1 and P_2 be convex polytopes in $\mathbb{R}^3$. Suppose that the maximum overlap of P_1 and P_2 under rigid motion is at least $\lambda \cdot \max\{|P_1|, |P_2|\}$ for some given constant $\lambda \in (0, 1]$. Given any $\varepsilon \in (0, 1/2)$, Algorithm 1 runs in $O(\varepsilon^{-3}\lambda^{-6} n \log^{3.5} n)$ time with probability at least $1 - n^{-O(1)}$ and returns a $(1-\varepsilon)$-approximate maximum overlap of P_1 and P_2 under rigid motion.*

Proof. The solution quality of algorithm 1 is guaranteed by Lemma 7. We analyze its running time as follows. First, it takes $O(n/\varepsilon)$ time to compute the ellipsoids $\mathcal{E}(P_1)$ and $\mathcal{E}(P_2)$. The remaining time spent by Algorithm 1 is $|S_\beta| \cdot |S_\alpha| \cdot |S_\gamma| \cdot n \log^{3.5} n$ with high probability. Thus, it suffices to bound $|S_\beta| \cdot |S_\alpha| \cdot |S_\gamma|$, which is $O\left(\varepsilon^{-3} \cdot |I_\beta||I_\alpha||I_\gamma| \cdot (\Delta_\beta \Delta_\alpha \Delta_\gamma)^{-1}\right)$.

Suppose that $2^{13}3^5 a_{\min} > \lambda^2 c_{\min}/\sqrt{2}$. Then $I_\xi = [-\pi, \pi]$ for $\xi \in \{\beta, \alpha, \gamma\}$. The assumption of $2^{13}3^5 a_{\min} > \lambda^2 c_{\min}/\sqrt{2}$ implies that $a_{\min}$, $b_{\min}$, and $c_{\min}$ are within constant factors of each other. Therefore, $\Delta_\beta \Delta_\alpha \Delta_\gamma = \Theta(1)$, which implies that $|S_\beta| \cdot |S_\alpha| \cdot |S_\gamma| = O(\varepsilon^{-3})$. Thus, the remaining time spent by Algorithm 1 is $O(\varepsilon^{-3} n \log^{3.5} n)$.

Suppose that $2^{13}3^5 a_{\min} \leqslant \lambda^2 c_{\min}/\sqrt{2}$. Then $|I_\beta| = O(a_{\min}/(\lambda^2 c_{\min}))$ and $\Delta_\beta = \Theta(a_{\min}/c_{\min})$, so $|I_\beta|/\Delta_\beta = O(\lambda^{-2})$. By definition, $|I_\alpha| = O(b_{\min}/(\lambda^2 c_{\min}))$ and $\Delta_\alpha = \Theta(b_{\min}/c_{\min})$, so $|I_\alpha|/\Delta_\alpha = O(\lambda^{-2})$. Similarly, $|I_\gamma| = O(a_{\min}/(\lambda^2 b_{\min}))$ and $\Delta_\gamma = \Theta(a_{\min}/b_{\min})$ by definition, which implies that $|I_\gamma|/\Delta_\gamma = O(\lambda^{-2})$. ◀

References

1. H.-K. Ahn, P. Brass, O. Cheong, H.-S. Na, C.-S. Shin, and A. Vigneron. Inscribing an axially symmetric polygon and other approximation algorithms for planar convex sets. *Comput. Geom. Theory and Appl.*, 33 (2006), 152–164.
2. H.-K. Ahn, S.-W. Cheng, and I. Reinbacher. Maximum overlap of convex polytopes under translation. *Proc. Internat. Sympos. Alg. and Comput.*, 2010, 97–108.
3. H.-K. Ahn, O. Cheong, C.-D. Park, C.-S. Shin, and A. Vigneron. Maximizing the overlap of two planar convex sets under rigid motions. *Comput. Geom. Theory and Appl.*, 37 (2007), 3–15.
4. H.-K. Ahn, P. Brass, and C.-S. Shin. Maximum overlap and minimum convex hull of two convex polyhedra under translation. *Comput. Geom. Theory and Appl.*, 40 (2008), 171–177.
5. M. de Berg, O. Cheong, O. Devillers, M. van Kreveld, and M. Teillaud. Computing the Maximum Overlap of Two Convex Polygons under Translations. *Theory of Comput. Syst.*, 31 (1998), 613–628.
6. H.J.A.M. Heijmans and A.V. Tuzikov. Similarity and symmetry measures for convex shapes using Minkowski addition. *IEEE Trans. PAMI*, 20 (1998), 980–993.
7. D.H. Lehmar. Approximations to the area of an n-dimensional ellipsoid. *Canadian J. Mathematics*, 2 (1950), 267–282.
8. F. Meyer and P. Bouthemy. Region-based tracking in an image sequence. *Proc. European Conf. Computer Vision*, 1992, 476–484.
9. M.A. Perles and G.C. Shephard. Euler-type relations for convex polytopes. *Mathematica Scandinavica*, 21 (1967), 199–218.
10. G.C. Shephard. Euler-type relations for convex polytopes. *Proc. London Mathematical Society*, 2 (1968), 597–606.
11. M. J. Todd and E. A. Yıldırım. On Khachiyan's algorithm for the computation of minimum volume enclosing ellipsoids. *Discr. Appl. Math.*, 155 (2007), 1731–1744.
12. R.C. Veltkamp and M. Hagedoorn. State of the art in shape matching. *Principles of Visual Information Retrieval*, (ed. M. Lew), Springer, 2001, 87–119.
13. A. Vigneron. Geometric optimization and sums of algebraic functions. *Proc. ACM–SIAM Sympos. Alg.*, 2010, 906–917.

Minimum Enclosing Circle with Few Extra Variables

Minati De[1], Subhas C. Nandy[1], and Sasanka Roy[2]

1 **Indian Statistical Institute, Kolkata - 700108, India**
{minati_r,nandysc}@isical.ac.in
2 **Chennai Mathematical Institute, Chennai - 603103, India**
sasanka@cmi.ac.in

Abstract

Asano et al. [JoCG 2011] proposed an open problem of computing the minimum enclosing circle of a set of n points in $\mathbb{R}^2$ given in a read-only array in sub-quadratic time. We show that Megiddo's prune and search algorithm for computing the minimum radius circle enclosing the given points can be tailored to work in a read-only environment in $O(n^{1+\epsilon})$ time using $O(\log n)$ extra space, where ϵ is a positive constant less than 1. As a warm-up, we first solve the same problem in an *in-place* setup in linear time with $O(1)$ extra space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Minimum enclosing circle, space-efficient algorithm, prune-and-search

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.510

1 Introduction

The minimum enclosing circle (MEC) for a set of points P is defined to be a circle of minimum radius that encloses all the points in P. The problem of finding the minimum enclosing circle has several vital applications. One such example is in planning the location of placing a shared facility like a hospital, gas station, or sensor devices etc. The center of the minimum enclosing circle will be the desired location for placing the facility. In the location theory community, this type of problem is known as the 1-center problem. It is first proposed by Sylvester in the year 1857 [15], and it asks for the location of a single facility that minimizes the distance (in some chosen metric) of the farthest demand point from the facility. Thus the problem we are considering is the Euclidean version of the 1-center problem. Elzinga et al. with their work [8] paved the way for solving minimax problems with elementary geometry, and proposed an $O(n^2)$ time algorithm for the Euclidean 1-center problem for a point set P, where $|P| = n$. Note that, (i) the MEC for the point set P is the same as the MEC for the convex hull of P (denoted by $CH(P)$), (ii) the center of the MEC of P is either on the mid-point of the diameter of $CH(P)$ or one of the vertices of the farthest point Voronoi diagram of P (denoted by $FVD(P)$), and (iii) $FVD(P) = FVD(CH(P))$. Since both computing $CH(P)$ and $FVD(P)$ need $O(n \log n)$ time [14], we have an $O(n \log n)$ time algorithm for computing the MEC of the point set P. The best known result for computing the MEC is an $O(n)$ time algorithm proposed by Megiddo [10]. Later Welzl [16] proposed an easy to implement randomized algorithm for computing the MEC that runs in expected $O(n)$ time. For the weighted version of the MEC problem, the best-known result is also by Megiddo [11] that runs in $O(n(\log n)^3(\log \log n)^2)$ time using the parametric search [9]. Later,

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 510–521

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

Megiddo and Zemel [12] proposed an $O(n \log n)$ time randomized algorithm for this problem that does not use parametric search. All of these algorithms use $O(n)$ extra work-space.

Recently, Asano et al. [2] proposed an $O(n^2)$ time and $O(1)$ extra-space algorithm for computing the vertices of the farthest point Voronoi diagram of P, where the points in the set P are given in a read-only array. Needless to say, the same time complexity holds for computing the minimum enclosing circle. In the same paper they mentioned the possibility of finding the minimum enclosing circle in sub-quadratic time in read-only setup with sub-linear work-space as an open problem. We answer this question affirmatively as stated below.

Our results: In this paper, we propose an algorithm for computing the minimum enclosing circle of a given set of n points in a read-only array. The time and extra space required for this algorithm are $O(n^{1+\epsilon})$ and $O(\log n)$ respectively, where $\sqrt{\frac{\log \log n}{\log n}} < \epsilon < 1$. As a warm-up, we first propose an algorithm for the same problem in an in-place model where swapping elements in the array is permissible. This needs $O(n)$ time and $O(1)$ extra-space. This algorithm is invoked in our proposed algorithm in the read-only setup.

Related works: If a set of n real values are given in an array, then the problem of computing the median in an in-place environment can be solved using at most $3n$ comparisons with $O(1)$ extra space [6]. If the array is read-only (i.e., swapping two values in the input array is prohibited) then the problem can be solved in $O(n^{1+\epsilon})$ time with $O(\frac{1}{\epsilon})$ space, where ϵ is a small (< 1) positive number to be fixed prior to the execution [13]. Chan [7] has shown that if S extra-bits are given in addition to the input array, then a lower bound on the expected time complexity of finding the median is $\Omega(n \log \log_S n)$. A lower bound on the deterministic time complexity for the same problem is $\Omega(n \log^* \frac{n}{S} + n \log_S n)$, where S extra-words are given as work space [7]. An important work in a different direction is the in-place algorithm for the linear programming problem with two variables, which can be solved in $O(n)$ time with $O(1)$ extra space, where n is the number of constraints [5]. This can be used to design a prune-and-search algorithm for finding the center of the minimum enclosing circle of the point set P where the center is constrained to lie on a given straight line [10].

Low-memory algorithms have many advantages compared to traditional algorithms [5, 6]. As they use only a very small amount of extra-space during their execution, a larger part of the data can be kept in the faster memory. As a result, the algorithm becomes faster. Readers are referred to [3, 4] for the in-place algorithms of several other geometric optimization problems. For the geometric algorithms in the read-only setup, see [1, 2].

2 Overview of Megiddo's algorithm

Let $P[0, \ldots, n-1]$ be an array containing n points. We now describe Megiddo's linear time algorithm for the MEC problem for the points in P. Let π^* be the center of desired MEC. At each iteration, it identifies a pair of mutually perpendicular lines such that the quadrant in which π^* lies can be identified, and a constant fraction of points in P can be deleted.

Algorithm 1: MEC(P)

Input: An array $P[1, \ldots, n]$ of points in $\mathbb{R}^2$.
Output: The center m^* of the minimum enclosing circle of the points in P.
while $|P| \geq 16$ **do**
  $P = PRUNE(P)$
(* Finally, when $|P| < 16$ *) compute the minimum enclosing circle in brute force manner.

Algorithm 2: PRUNE(P)

Input: An array $P[1,\ldots,n]$ of points in $\mathbb{R}^2$.
Output: The set of points P after pruning.
Step 1: Arbitrarily pair up the points in P. Let $(P[2i], P[2i+1]), i = 0, 1, \ldots, \lfloor \frac{n}{2} \rfloor)$ be the aforesaid pairs;
Step 2: Let L_i denote the bisector of the pair of points $(P[2i], P[2i+1])$, and $\alpha(L_i)$ denote the angle of L_i with the x-axis. Compute the median μ of $\{\alpha(L_i), i = 0, 1, \ldots, \lfloor \frac{n}{2} \rfloor\}$;
Step 3: Arbitrarily pair up (L_i, L_j) where $\alpha(L_i) \leq \mu$ and $\alpha(L_j) \geq \mu$. Let M be the set of these $\lfloor \frac{n}{4} \rfloor$ pairs of lines;
We split M into two subsets M_P and M_I, where $M_P = \{(L_i, L_j) | \alpha(L_i) = \alpha(L_j) = \mu\}$ (* parallel line-pairs *) and $M_I = \{(L_i, L_j) | \alpha(L_i) \neq \alpha(L_j)\}$ (* intersecting line-pairs *);
for *each pair* $(L_i, L_j) \in M_P$ **do**
  compute $y_{ij} = \frac{d_i + d_j}{2}$, where d_i = distance of L_i from the line $y = \mu x$
for *each pair* $(L_i, L_j) \in M_I$ **do**
  Let a_{ij} = point of intersection of L_i & L_j, and b_{ij} = projection of a_{ij} on $y = \mu x$. Compute
  y_{ij} = signed distance of the pair of points (a_{ij}, b_{ij}), and
  x_{ij} = signed distance of b_{ij} from the origin;
Next, compute the median y_m of the y_{ij} values corresponding to all the pairs in M;
Step 4: Consider the line $\mathcal{L}_H : y = \mu x + y_m \sqrt{\mu^2 + 1}$, which is parallel to $y = \mu x$ and at a distance y_m from $y = \mu x$;
Compute the center π of the constrained minimum enclosing circle whose center lies on $\mathcal{L}_H$ using Algorithm *Constrained_MEC*$(P, \mathcal{L}_H)$;
Step 5: (* Decide in which side of $\mathcal{L}_H$ the center π^* of the unconstrained MEC lies *);
Let Q be the set of points in P that are farthest from π;
if $|Q| = 1$ **then** π^* and the only point $p_i \in Q$ lie in the same side of $\mathcal{L}_H$;
if $|Q| \geq 2$ **then**
  if all the members of Q lie in the same side of $\mathcal{L}_H$, **then** π^* will also lie in that side of $\mathcal{L}_H$;
  otherwise (* we need to check whether the convex polygon formed by the points in Q contain π or not as follows *)
  Let Q_1 and Q_2 be two subsets of Q that lie in two different sides of $\mathcal{L}_H$ respectively;
  $Q_1 \cup Q_2 = Q$. Find two points $p_i, p_j \in Q_1$ that make maximum and minimum angles with $\mathcal{L}_H$ with center at π in anticlockwise direction. Now consider each point $q \in Q_2$ and test whether $\pi \in \Delta p_i q p_j$.
  Similarly, find $p_k, p_\ell \in Q_2$ that make maximum and minimum angles with $\mathcal{L}_H$ with center at π in clockwise direction. Consider each point $q' \in Q_1$ and test whether $\pi \in \Delta p_k q' p_\ell$;
  If any one of these triangles contain π, then the convex polygon Q contains π. Here the algorithm stops reporting $\pi^* = \pi$.
  Otherwise, either (p_i, p_k) or (p_j, p_ℓ) define the diagonal (farthest pair of points) in Q. Let q be the mid-point of the diagonal. Here, π^* and q will lie in the same side of $\mathcal{L}_H$;
Step 6: Let $M'_I = \{(L_i, L_j) \in M_I |$ a_{ij} and π^* lie in the different sides of the line $\mathcal{L}_H$ $\}$;
Compute the median x_m of x_{ij}-values for the line-pairs in M'_I. Define a line $\mathcal{L}_V$ perpendicular to $y = \mu x$ and passing through a point on $y = \mu x$ at a distance x_m from the origin;
Execute Algorithm *Constrained_MEC*$(P, \mathcal{L}_V)$ and decide in which side of $\mathcal{L}_V$ the point π^* lies as in Step 5;
From now onwards, we will denote $\mathcal{L}_H$ and $\mathcal{L}_V$ as horizontal and vertical lines respectively;
Without loss of generality, assume that π^* lies in the top-left quadrant;
Step 7: (* Pruning step *)
for *all the members* $(L_i, L_j) \in M_I$ *whose points of intersection* (a_{ij}) *lie in the bottom-right quadrant* **do**
  Let $\alpha(L_i) \leq \mu$ and L_i be defined by the pair of points $(P[2i], P[2i+1])$;
  Discard one of $P[2i]$ and $P[2i+1]$ which is top-left to other one from P;
for *all the members* $(L_i, L_j) \in M_P$ *whose* $y_{ij} \leq y_m$ **do**
  Let L_i be below $\mathcal{L}_H$, and L_i be defined by a pair of points $[P[2i], P[2i+1])$;
  Discard either $P[2i]$ or $P[2i+1]$ depending on which one lies above L_i;
Step 9: return P (* Now P denotes the set of points after pruning *).

Algorithm 3: Constrained_MEC(P, L)

Input: An array $P[1, \ldots, n]$ of points in $\mathbb{R}^2$, and a line L (* assumed to be vertical *).
Output: The center m^* of the minimum enclosing circle of the points in P on the line L.
Step 1:
while $|P| \geq 3$ **do**
  Step 1.1: Arbitrarily pair up the points in P. Let $(P[2i], P[2i+1]), i = 0, 1, \ldots, \lfloor \frac{n}{2} \rfloor$ be the aforesaid pairs;
  Step 1.2: Let ℓ_i denote the perpendicular bisector of the pair of points $(P[2i], P[2i+1])$, $i = 0, 1, \ldots \lfloor \frac{n}{2} \rfloor$. Let ℓ_i intersect L at a point q_i. and $Q = \{q_i, i = 0, 1, \ldots, \lfloor \frac{n}{2} \rfloor\}$;
  Step 1.3: Compute the median m of the y-coordinate of the members of Q;
  Step 1.4: (* Test on which side (above or below) of m the center m^* of the constrained MEC lies (i.e., whether $m^* < m$ or $m^* > m$) as follows: *)
  Identify the point(s) $F \subset P$ that is/are farthest from m;
  if *the projection of all the members in F on L are in different sides of m* **then**
    return $m^* = m$ (* center of the constrained minimum enclosing circle on the line L *)
  else
    (* i.e., the projection of all the members in F on L are in the same side (above or below) of m *) m^* lies in that side of m on the line L
  Step 1.5: Without loss of generality, assume, that $m^* > m$. Then for each bisector line $\ell_{p,q}$ (defined by the point-pair $p, q \in P$) that cuts the line L below the point m, we can delete one point among p and q from P such that the said point and the point m lie in the same side of $\ell_{p,q}$;
Step 2: (* the case when |P|=2, *)
if *the perpendicular bisector of the members of P intersects L* **then**
  return the point of intersection as m^*;
else
  (* the perpendicular bisector of the members of P is parallel with L *)
  return m^* = projection of the farthest point of P on L.

The correctness of the algorithm is given in [10]. An iteration of the procedure *PRUNE* with the set of points P needs $O(|P|)$ time, and it deletes at least $\lfloor \frac{|P|}{16} \rfloor$ points from P. Thus, Megiddo's algorithm for the MEC problem executes the procedure *PRUNE* at most $O(\log n)$ times, and its total running time is $O(n)$ time using $O(n)$ extra space.

3 In-place implementation of MEC

In this section, we will show that Megiddo's algorithm (stated in the Section 2) can be made in-place with the same time complexity. It is to be noted that we may succeed in making all the steps in-place separately but there may be problems while integrating them together. For an example, one can easily be able to make the *Constrained_MEC* (Step 4 of the procedure *PRUNE*) in-place (as *2D linear programming* can be solved in an in-place manner in linear time [5]), but one will have to assure that after this, one will be able to figure out the chosen pair of bisectors satisfying the condition mentioned in Step 3 of the procedure *PRUNE*, as this will be required in the Step 6 of the same procedure. We will ensure this integration. We will extensively use the fact that the median of a set of n numbers stored in an array of size n can be computed in an in-place manner in $O(n)$ time using $O(1)$ extra-space [6].

In Step 2 of the procedure *PRUNE* we can compute the median angle μ in an in-place manner. Note that we do not have to store the $\{L_i, i = 0, 1, \ldots \lfloor \frac{n}{2} \rfloor\}$ as one can compute them on demand with the knowledge of $(P[2i], P[2i+1])$.

Step 3 of the procedure *PRUNE* can be made in-place in $O(n)$ time and $O(1)$ extra space as

follows: identify $\lfloor \frac{n}{4} \rfloor$ pairs (L_i, L_j) ($\alpha(L_i) \leq \mu$ and $\alpha(L_j) \geq \mu$), and for each pair accumulate the tuple of four points $(P[2i], P[2i+1], P[2j], P[2j+1])$ in consecutive locations of the array. Note that this consecutive arrangement will help in computing x_{ij} and y_{ij} for L_i and L_j (see Step 3 of Procedure *PRUNE*) on the fly. So, we maintain the following *invariant*

▶ Invariant 1. (i) During the execution of Steps 3-6 of the procedure *PRUNE*, the pair of points (p, q) of P defining L_i (their perpendicular bisector), for each $i = 0, 1, \ldots \lfloor \frac{n}{2} \rfloor$ will remain in consecutive locations of the input array P.

(ii) During the execution of Steps 4-6 of the procedure *PRUNE*, the tuple of points (p, q, r, s) of P, defining the y_{ij}-value for two bisectors (L_i, L_j) ((p, q) defining L_i and (r, s) defining L_j) that satisfy $\alpha(L_i) \leq \mu$ and $\alpha(L_j) \geq \mu$, will remain in consecutive locations of the input array P.

We store the number of input points in a variable n, and use a variable ν to denote the current size of the array P. In each iteration of the Algorithm MEC, after the (pruning) Step 7 of the procedure *PRUNE*, the deleted points are moved at the end of the array, and ν is updated to the number of non-deleted points. We have already shown that Steps 1-3 can be made in-place. In the next subsection, we show that Steps 4-6 can also be made in-place satisfying *invariant* 1 (see Lemma 3). Thus, we have the following result.

▶ **Theorem 1.** *Minimum enclosing circle of a set of n points in $\mathbb{R}^2$ can be computed in an in-place manner in $O(n)$ time with $O(1)$ extra work-space.*

3.1 In-place implementation of *Constrained_MEC*

In a particular iteration of the algorithm MEC, we have all non-deleted points stored in consecutive locations of the array P starting from its leftmost cell. In Step 4 of the procedure *PRUNE*, we use the procedure *Constrained_MEC* to compute the center m^* of the minimum enclosing circle for these points where m^* is constrained to lie on the given line L. Without loss of generality, let us assume that L is a vertical line. A straight forward way to implement this procedure in an in-place manner without maintaining *Invariant* 1 is as follows.

Find the median point m on the line L among the points of intersection of the lines ℓ_i and L for $i = 1, 2, \ldots, \frac{n}{2}$ in an in-place manner using the algorithm given in [6], where the points of intersection are computed on the fly. This needs $O(n)$ time. Next, inspect all the points to decide whether m^* is above or below m as follows. Let F denote the set of points in P which are farthest from m.

- If the projection of the members in F on the line L lie in both the sides of m, then $m^* = m$.
- If the projection of all the members in F on the line L lie in the same side (above or below) of m, then m^* lies in that side of m on the line L.

If $m^* = m$ then the iteration in *Constrained_MEC* stops; otherwise the following pruning step is executed for the next iteration. Without loss of generality, let m^* be above m. We again scan each $\ell_i = (P[2i], P[2i+1])$ and compute its intersection with L. If it is below m, then we delete the one which is on the same side of m with respect to the bisector line ℓ_i. As we have $\frac{n}{4}$ intersection points below m, we can delete (i.e., move at the end of the array) $\frac{n}{4}$ points from P. The case where m^* is below m can be handled similarly. The entire procedure *Constrained_MEC* needs $O(n)$ time and $O(1)$ extra space, but after an iteration Invariant 1 will not remain valid.

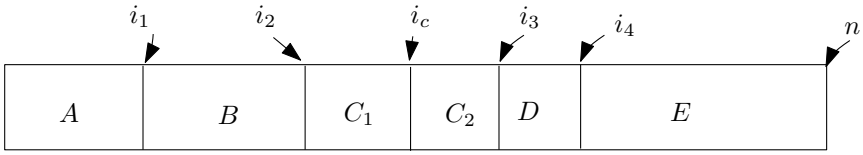


Figure 1 Block Partition of the Array P

To resolve this problem, we do the following. During the execution of *Constrained_MEC*, if a point is deleted from a tuple (p, q, r, s) in an iteration, it is considered to be *invalid* from next iteration onwards. We partition the array P containing all the points into five blocks namely A, B, C, D and E and use four index variables i_1, i_2, i_3 and i_4 to mark the ending of the first four blocks (see Figure 1). Block A consists of those tuple (p, q, r, s) whose four points are *invalid*. The block B signifies all those tuples containing three *invalid* points. Similarly, block C, D contain tuples with two and one *invalid* point(s) respectively. Block E contains all tuples with no *invalid* point. We further partition the block C into two sub-blocks C_1 and C_2 respectively. The tuples with first two *invalid* points are kept in C_1 and the tuples with first and third *invalid* points are stored in C_2. If a tuple has *invalid* points in second (resp. fourth) position, then these are swapped to first (resp. third) position. We use an index variable i_c to mark the partition between C_1 and C_2. All the *invalid* points in a tuple belonging to block B and D are kept at the beginning of that tuple. In other words, during the entire execution of *Constrained_MEC*, we maintain the following invariant along with the *Invariant* 1.

▶ Invariant 2. The tuples with zero, one, two, three and four valid point(s) will be in the block A, B, C, D and E, respectively as mentioned above.

Now, we need (i) to form the bisector lines $\{\ell_i, i = 1, 2, \ldots \lfloor \frac{n}{2} \rfloor\}$, and then (ii) to find the median m of the points of intersection of these bisector lines with L in an in-place manner using the algorithm given in [6]. If we form these bisector lines with two consecutive valid points in the array P, then the Invariant 1 may not be maintained since (i) during the median finding ℓ_i's need to be swapped, and (ii) the points in a tuple may contribute to different ℓ_i's.

Here three important things need to be mentioned:

Formation of ℓ_i: Each tuple in block B contains only one *valid* point. Thus, we pair up two tuples to form one bisector line ℓ_i in Step 1 of the algorithm *Constrained_MEC*. Thus, we will have $\lfloor \frac{1}{2}(\frac{i_2 - i_1}{4}) \rfloor$ bisectors. Let's denote these set of bisectors by $\mathcal{L}_1$.
Similarly, C_1 and C_2 will produce $\frac{i_c - i_2}{4}$ and $\frac{i_3 - i_c}{4}$ bisector lines respectively, and these are denoted as $\mathcal{L}_2$ and $\mathcal{L}_3$ respectively.
In block D, each tuple (p, q, r, s) contains three *valid* points and the *invalid* point is p. In each of these tuples, we consider the pair of points (r, s) to form a bisector line. Let us denote this set of bisectors by $\mathcal{L}_4$, and the number of bisectors in this set is $\frac{i_4 - i_3}{4}$.
Next we consider each pair of consecutive tuples (p, q, r, s) and (p', q', r', s') in block D, and define a bisector line with the *valid* point-pair (q, q'). Thus we get $\lfloor \frac{1}{2}(\frac{i_4 - i_3}{4}) \rfloor$ such bisectors, and name this set $\mathcal{L}_5$.
From each tuple (p, q, r, s) in block E, we get two bisectors. Here we form two sets of bisectors, namely $\mathcal{L}_6$ and $\mathcal{L}_7$. $\mathcal{L}_6$ is formed with (p, q) of each tuple in block E, and $\mathcal{L}_7$ is formed with (r, s) of each tuple in block E. Each of these sets contains $\lfloor \frac{n - i_4}{4} \rfloor$ bisectors. Thus, we have seven sets of bisectors, namely $\mathcal{L}_i, i = 1, 2, \ldots, 7$.

Computing median: We compute the median of the points of intersection of the lines in each set of bisector lines $\mathcal{L}_i$ with L separately. We use m_i to denote the median for i-th

set. During the execution of in-place median finding algorithm of [6], if a pair of lines $\ell_i, \ell_j \in \mathcal{L}_k$ are swapped then the corresponding entire tuple(s) are swapped. Thus, the tuples are not broken for computing the median and both the Invariants 1 and 2 are maintained.

Pruning step: We take two variables m' and m'' to store two points on the line L such that the desired center m^* of the minimum enclosing circle of P on L satisfies $m' \leq m^* \leq m''$. We initialize $m' = -\infty$ and $m'' = \infty$. Now, we consider each $m_i, i = 1, 2, \ldots, 7$ separately; if m^* is above m_i and $m' < m_i$, then m' is set to m_i. If m^* is below m_i and $m'' > m_i$ then m'' is set to m_i.

We now prune points by considering the intersection of the bisector lines in $\cup_{i=1}^{7}\mathcal{L}_i$ with L. If a bisector line $\ell = (p, q) \in \cup_{i=1}^{7}\mathcal{L}_i$ intersects L in the interval $[m', m'']$ then none of p, q becomes *invalid*; otherwise one of the points p or q becomes *invalid* as mentioned in Step 4 of the Procedure *Constrained_MEC*.

While considering the bisector lines in $\mathcal{L}_1$, a tuple in the block B may be moved to block A by swapping that tuple with the first tuple of block B and incrementing i_1 by 4.

While considering a bisector line $\ell \in \mathcal{L}_2 \cup \mathcal{L}_3$, if any one of its participating points is deleted then the corresponding tuple is moved to block B by executing one or two swap of tuple and incrementing i_2 by 4.

Note that, the bisector lines in $\mathcal{L}_4$ and $\mathcal{L}_5$ are to be considered simultaneously. For a pair of tuple $(p, q, r, s), (p', q', r', s') \in D$, we test the bisector lines $\ell = (q, q') \in \mathcal{L}_4$ and $\ell' = (r, s) \in \mathcal{L}_4$ and $\ell'' = (r', s') \in \mathcal{L}_5$ with $[m', m'']$. This may cause deletion of one or two points from (p, q, r, s) (resp. (p', q', r', s')). If for the tuple (p, q, r, s),

- none of the points becomes *invalid*, it will remain in the set D;
- if only q becomes invalid, it is moved to C_1 by two swaps of tuples; necessary adjustments of i_c and i_3 need to be done;
- if r or s only becomes *invalid*, it is moved to C_2 (with a swap of r and s if necessary), and adjustment of i_3 is done;
- if q and r both become *invalid*, it is moved to B with necessary adjustment of i_2, i_3;
- if q and s both become *invalid*, then it is moved to B (with swap among r and s) and necessary adjustment of i_2, i_3 need to be done.

The same set of actions may be necessary for the tuple (p', q', r', s') also.

Similarly, the bisector lines in $\mathcal{L}_6$ and $\mathcal{L}_7$ are considered simultaneously. For a tuple $(p, q, r, s) \in E$, $\ell = (p, q) \in \mathcal{L}_6$ and $\ell' = (r, s) \in \mathcal{L}_7$. Here none or one or two points from the tuple (p, q, r, s) may be deleted. Depending on that, it may reside in the same block or may be moved to block D or C_2. The necessary intra-block movement can be done with one or two tuple-swap operation. Surely at most two swap operation inside the tuple may be required to satisfy Invariant 2.

3.1.0.1 Correctness and complexity results

▶ **Lemma 2.** *The above pruning steps ensure Invariants 1 and 2 and at least* $\frac{n}{4}$ *points become* invalid *after each iteration, where* n *is the number of valid points in* P *at the beginning of the iteration.*

Proof. The description of the pruning step justifies the first part of the lemma. For the second part, note that m_i (the median of the intersection points of the members in $\mathcal{L}_i$ with L) satisfies either $m_i \leq m'$ or $m_i \geq m''$. In both the cases, at least half of the lines in $\mathcal{L}_i$ intersect L outside the interval $[m', m'']$. Thus, the result follows. ◀

The correctness of the algorithm follows from the fact that after an iteration of the *Constrained_MEC*, the valid points can be easily identified using our proposed scheme of maintaining the points in five different blocks as mentioned in Invariant 2. It also helps in forming the bisector lines, and pruning of points maintaining Invariant 1. The second part of Lemma 2 justifies the following result.

▶ **Lemma 3.** *The Constrained_MEC can be computed in an in-place manner in $O(n)$ time with constant amount of extra space.*

4 When the memory is read-only

In this section, we show how one can compute the minimum enclosing circle efficiently for a set of points in $\mathbb{R}^2$ with $O(\log n)$ extra variables, when the input points are given in a read-only array P. Here again we will use the basic algorithm *MEC* of Megiddo as described in Section 2. As we are not allowed to move the deleted elements to one end, the main challenge in read-only memory for implementing Megiddo's algorithm is in detecting the *valid* points after pruning.

Long ago Munro and Raman [13] gave a space-time trade-off for median finding algorithms in read-only memory. Though we can not use their algorithm directly for median finding in our setup, we will use a similar idea. For ease of understanding, we will briefly describe the median finding algorithm of [13]. Next we will describe our approach for computing the minimum enclosing circle for the points in the array P.

4.1 Munro and Raman's median finding algorithm

Given a set of n points in $\mathbb{R}$ in a read-only array P, the algorithm of [13] is designed by using a set of procedures $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$, where procedure $\mathcal{A}_i$ finds the median by evoking the procedure $\mathcal{A}_{i-1}$ for $i \in \{1, 2, \ldots, k\}$. The procedures $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ are stated below.

Procedure $\mathcal{A}_0$: In the first iteration, after checking all the elements in P, it finds the largest element $p_{(1)}$ in linear time. In the second iteration it finds the second largest $p_{(2)}$ by checking only the elements which are less than $p_{(1)}$. Proceeding in this way, in the j-th iteration it finds the j-th largest element $p_{(j)}$ considering all the elements in P that are less than $p_{(j-1)}$. In order to get the median we need to proceed up to $j = \lfloor \frac{n}{2} \rfloor$. Thus, this simple median finding algorithm takes $O(n^2)$ time and $O(1)$ extra-space.

Procedure $\mathcal{A}_1$: It divides the array P into blocks of size $\sqrt{n}$ and in each block it finds the median using Procedure $\mathcal{A}_0$. After computing the median m of a block, it counts the number of elements in P that are smaller than m, denoted by $\rho(m)$, by checking all the elements in the array P. It maintains two best block medians m_1 and m_2, where $\rho(m_1) = \max\{\rho(m) | \rho(m) \leq \frac{n}{2}\}$, and $\rho(m_2) = \min\{\rho(m) | \rho(m) \geq \frac{n}{2}\}$. Thus, this iteration needs $O(n\sqrt{n})$ time.

After this iteration, all the elements $P[i]$ satisfying $P[i] < m_1$ or $P[i] > m_2$ are marked as *invalid*. This does not need any mark bit; only one needs to remember m_1 and m_2. In the next iteration we again consider same set of blocks, and compute the median ignoring the *invalid* elements.

Since, in each iteration $\frac{1}{4}$ fraction of the existing *valid* elements are marked *invalid*, we need at most $O(\log n)$ iterations to find the median μ. Thus the time complexity of this procedure is $O(n\sqrt{n} \log n)$.

Procedure $\mathcal{A}_2$: It divides the whole array into $n^{1/3}$ blocks each of size $n^{2/3}$, and computes the block median using the procedure $\mathcal{A}_1$. Thus, the overall time complexity of this procedure for computing the median is $n^{1+\frac{1}{3}} \log^2 n$.

Proceeding in this way, the time complexity of the procedure $\mathcal{A}_k$ will be $O(n^{(1+\frac{1}{k+1})} \log^k n)$. As it needs a stack of depth k for the recursive evoking of $\mathcal{A}_{k-1}$, $\mathcal{A}_{k-2}$, ..., $\mathcal{A}_0$, the space complexity of this algorithm is $O(k)$.

Setting $\epsilon = \frac{1}{k+1}$, gives the running time as $O(\frac{n^{1+\epsilon} \log^{\frac{1}{\epsilon}} n}{\log n})$. If we choose $n^\epsilon = \log^{\frac{1}{\epsilon}} n$, then ϵ will be $\sqrt{\frac{\log \log n}{\log n}}$, and this will give the running time $O(\frac{n^{1+2\epsilon}}{\log n})$, which is of $O(n^{1+2\epsilon})$. So, the general result is as follows:

▶ Result 1. For a set of n points in $\mathbb{R}$ given in a read-only memory, the median can be found in $O(n^{1+\epsilon})$ time with $O(\frac{1}{\epsilon})$ extra-space, where $2\sqrt{\frac{\log \log n}{\log n}} \leq \epsilon < 1$.

4.2 Algorithm MEC in read-only setup

Given a set of n points in $\mathbb{R}^2$ in a read-only array P of size n, our objective is to compute the minimum enclosing circle of the points in P using $O(\log n)$ extra space. We first show how one can compute *Constrained_MEC* when the input array is read-only using $O(\log n)$ extra variables. Next, we use this algorithm along with another $O(\log n)$ space to compute the center of the unconstrained minimum enclosing circle.

4.2.1 *Constrained_MEC* in read-only setup

We first note that at each iteration of the procedure *Constrained_MEC* at least $\frac{1}{4}|P|$ points in P are pruned (marked *invalid*). Thus, the number of iterations executed in the procedure *Constrained_MEC* is at most $O(\log |P|)$.

We use an array M each element of which can store a real number, and an array D each element of which is a bit. Both the arrays are of size $O(\log |P|)$. After each iteration of the read-only algorithm, it needs to remember the median m among the points of intersection of the bisector lines on the line L, and the direction in which we need to proceed from m to reach the constrained center m^*. So, after executing the i-th iteration, we store m at $M[i]$; $D[i]$ will contain 0 or 1 depending on whether $m^* > m$ or $m^* < m$.

We now explain the i-th iteration assuming that $(i-1)$ iterations are over. Here we need to pair-up points in P in such a way that all the *invalid* elements up to the $(i-1)$-th iteration can be ignored correctly. We use one more array $IndexP$ of size $\log |P|$. At the beginning of this iteration all the elements in this array are initialized with -1.

Note that, we have no space to store the mark bit for the *invalid* points in the array P. Thus, we use the *compute in lieu of store* paradigm, or in other words, we check whether a point is *valid* at the i-th iteration, by testing its validity in all the $t = 1, 2, \ldots, i-1$ *levels* (previous iterations).

We start scanning the input array P from the left, and identify the points that are tested as *valid* in the t-th level for all $t = 1, 2, \ldots, i-1$. As in the in-place version of the *Constrained_MEC* algorithm, here also we pair up these valid points for computing the bisector lines. Here we notice the following fact:

Suppose in the $(i-1)$-th iteration (p, q) form a pair, and p is observed as *invalid*. While

executing the i-th iteration, we again need to check whether p was *valid* in the $i-1$-th iteration since it was not marked. Now, during this checking if we use a different point q' ($\neq q$) to form a pair with p, it may be observed *valid*. So, during the checking in the i-th iteration, (p, q) should be paired at the $(i-1)$-th level.

Thus, our pairing scheme for points should be such that it must satisfy the following invariant.

▶ Invariant 3. If (i) two points $p, q \in P$ form a point-pair at the t-th level in the j-th iteration, and (ii) both of them remain *valid* up to k-th iteration where $k > j$, then p, q will also form a point-pair at the t-th level of the the k-th iteration.

Pairing scheme: We consider the point-pairs $(P[2\alpha], P[2\alpha+1])$, $\alpha = 0, 1, \ldots, \lfloor \frac{n}{2} \rfloor$ in order. For each pair, we compute their bisector ℓ_α, and perform the level 1 test using $M[1]$ and $D[1]$ to see whether both of them remains *valid* at iteration 1. In other words, we observe where the line ℓ_α intersects the vertical line $x = M[1]$, and then use $D[1]$ to check whether any one of the points $P[2\alpha]$ and $P[2\alpha+1]$ becomes *invalid* or both of them remain *valid*. If the test succeeds, we perform level 2 test for ℓ_α by using $M[2]$ and $D[2]$. We proceed similarly until (i) we reach up to i-th level and both the points remain *valid* at all the levels, or (ii) one of these points is marked *invalid* at some level, say j ($< i-1$). In Case (i), the point pair $(P[2\alpha], P[2\alpha+1])$ participates in computing the median value m_i. In case (ii), suppose $P[2\alpha]$ remains *valid* and $P[2\alpha+1]$ becomes *invalid*. Here two situations need to be considered depending on the value of $IndexP[j]$. If $IndexP[j] = -1$ (no point is stored in $IndexP[j]$), we store 2α or $2\alpha+1$ in $IndexP[j]$ depending on whether $P[2\alpha]$ or $P[2\alpha+1]$ remains *valid* at level j. If $IndexP[j] = \beta (\neq -1)$ (index of a *valid* point), we form a pair $(P[2\alpha], P[\beta])$ and proceed to check starting from $j+1$-th level (i.e., using $M[j+1]$ and $D[j+1]$) onwards until it reaches the i-th level or one of them is marked *invalid* in some level between j and i. Both the situations are handled in a manner similar to Cases (i) and (ii) as stated above.

▶ **Lemma 4.** *Invariant 3 is maintained throughout the execution.*

Proof. Follows from the fact that the tests for the points in P at different levels $t = 1, 2, \ldots, i-2$ at both the $(i-1)$-th and i-th iterations are the same. At the $(i-1)$-th level of the $(i-1)$-th iteration, we compute m_{i-1} and D_{i-1} with the *valid* points. At the $(i-1)$-th level of the i-th iteration, we prune points that were tested *valid* at the $(i-1)$-th iteration using M_{i-1} and D_{i-1}. ◀

▶ Observation 1. At the end of the i-th iteration,

(i) Some cells of the $IndexP$ array may contain valid indices ($\neq -1$).
(ii) In particular, $IndexP[i-1]$ will either contain -1 or it will contain the index of some point β in P that has participated in computing m_{i-1} (i.e., remained *valid* up to level $i-1$).
(iii) If in this iteration $IndexP[i-1] = \beta$ (where β may be a valid index or -1), then at the end of all subsequent iterations j ($> i$) it will be observed that $IndexP[i-1] = \beta$.

Proof. Part (i) follows from the pairing scheme. Parts (ii) & (iii) follow from Lemma 4. ◀

▶ **Lemma 5.** *In the i-th iteration, the amortized time complexity for finding all valid pairs is $O(ni)$.*

Proof. Follows from the fact that each *valid* point in the i-th iteration has to qualify as a *valid* point in the tests of all the $i-1$ levels. For any other point the number of tests is at most $i-2$. ◀

The main task in the i-th iteration is to find the median of the points of intersection of all the valid pairs in that iteration with the given line L. We essentially use the median finding algorithm in [13] for this purpose. Notice that, in order to get each intersection point, we need to get a *valid* pair of points, which takes $O(i)$ time (see Lemma 5). Assuming L to be horizontal, the time required for finding the leftmost intersection point on L is $O(ni)$. Similarly, computing the second left-most intersection point needs another $O(ni)$ time. Proceeding similarly, the time complexity of the procedure $\mathcal{A}_0$ of [13] is $O(n^2i^2)$. Similarly, $\mathcal{A}_1$ takes $O(i^2n^{1+\frac{1}{2}}\log n)$ time, and so on. Finally, $\mathcal{A}_k$ takes $O(i^2n^{(1+\frac{1}{k+1})}\log^k n)$ time. Since we have chosen $k = \sqrt{\frac{\log n}{\log\log n}} < \log n$ for the median finding algorithm of [13], we need $O(\log n)$ space in total. Thus, we have the following result:

▶ **Lemma 6.** *The time complexity of the i-th iteration of Constrained_MEC is $O(i^2n^{(1+\frac{1}{k+1})}\log^k n)$, where $1 \leq k \leq \sqrt{\frac{\log n}{\log\log n}}$. The extra space required is $O(\log n)$.*

At the end of the $O(\log n)$ iterations, we could discard all the points except at most $|IndexP|+3$ points, where $|IndexP|$ is the number of cells in the array $IndexP$ that contain valid indices of P ($\neq -1$). This can be at most $O(\log n)$. We can further prune the points in the $IndexP$ array using the in-place algorithm for *Constrained_MEC* proposed in Section 3.1. Thus, we have the following result:

▶ **Lemma 7.** *The time complexity of Constrained_MEC is $O(n^{(1+\frac{1}{k+1})}\log^{k+3} n)$, where where $1 \leq k \leq \sqrt{\frac{\log n}{\log\log n}}$. Apart from the input array, it requires $O(\log n)$ extra space.*

Proof. By Lemma 6, the time complexity of the i-th iteration is $O(i^2n^{(1+\frac{1}{k+1})}\log^k n)$, where $i = 1, 2, \ldots, \log n$. Thus, the total time complexity of all the $O(\log n)$ iterations is $O(n^{(1+\frac{1}{k+1})}\log^{k+3} n)$. The extra time required by the in-place algorithm for considering all the entries in the array $IndexP$ is $O(\log n)$ (see Lemma 3), and it is subsumed by the time complexity of the iterative algorithm executed earlier.

The space complexity follows from the fact that the same set of arrays M, D, $IndexP$ and the stack for finding the median can be used for all the $\log n$ iterations, and each one is of size at most $O(\log n)$. ◀

4.2.2 Unconstrained MEC in a read-only setup

As earlier, we use the read-only variation of the *Constrained_MEC* algorithm (described in Subsection 4.2.1) for solving the unconstrained minimum enclosing circle problem. Here we need to maintain three more arrays $\mathcal{M}$, $\mathcal{D}$ and $\mathcal{I}$, each of size $O(\log n)$. $\mathcal{M}[i]$ contains the point of intersection of the vertical and horizontal lines used for pruning points at level i of the algorithm MEC; $\mathcal{D}[i]$ (a two bit space) indicates the quadrant in which the center of the MEC lies. The array $\mathcal{I}$ plays the role of the array $IndexP$ used for *Constrained_MEC*. It is shared by all the iterations of the algorithm.

While checking a point to be *valid* in any iteration of the procedure *Constrained_MEC* at the i-th iteration of the *MEC* algorithm, we first need to check whether it is pruned in any previous iteration of the algorithm *MEC*.

▶ **Theorem 8.** *The minimum enclosing circle of a set of n points in $\mathbb{R}^2$ given in a read-only array can be found in $O(n^{1+\epsilon})$ time and $O(\log n)$ space, where $\sqrt{\frac{\log\log n}{\log n}} < \epsilon < 1$.*

Proof. In the i-th iteration of the algorithm MEC, the time required for Steps 1-3 of the procedure $PRUNE$ is $O(i^2 n^{(1+\frac{1}{k+1})} \log^k n)$ (see the justifications of Lemma 6). In Step 4, the procedure *constrained_MEC* needs $O(n^{(1+\frac{1}{k+1})} \log^{k+3} n)$ time (see Lemma 7). Since the algorithm MEC consists of at most $O(\log n)$ iterations of the procedure $PRUNE$, the overall time complexity of the algorithm is $O(n^{(1+\frac{1}{k+1})} \log^{k+4} n)$. Substituting $\frac{\epsilon}{2} = \frac{1}{k+1}$ and then $n^{\frac{\epsilon}{2}} \geq \log^{4+\frac{1}{\epsilon}} n$, we have time complexity $O(n^{1+\epsilon})$, where ϵ satisfies $\sqrt{\frac{\log \log n}{\log n}} < \epsilon < 1$. ◀

5 Conclusion

In this paper, we propose a general prune-and-search technique in read-only memory which can be applied in other problems as well. Our in-place MEC as well as read-only MEC algorithm significantly improve the previously known best results. It will be worthy to study whether one can further improve the time-space complexity of MEC in a read-only setting.

References

1 T. Asano and B. Doerr. Memory-constrained algorithms for shortest path problem. In *CCCG*, 2011.
2 T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *JoCG*, 2(1):46–68, 2011.
3 P. Bose, A. Maheshwari, P. Morin, J. Morrison, M. H. M. Smid, and J. Vahrenhold. Space-efficient geometric divide-and-conquer algorithms. *Comput. Geom.*, 37(3):209–227, 2007.
4 H. Brönnimann, T. M. Chan, and E. Y. Chen. Towards in-place geometric algorithms and data structures. In *Symp. on Comput. Geom.*, pages 239–246, 2004.
5 H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. T. Toussaint. Space-efficient planar convex hull algorithms. *Theor. Comput. Sci.*, 321(1):25–40, 2004.
6 S. Carlsson and M. Sundström. Linear-time in-place selection in less than 3n comparisons. In *ISAAC*, pages 244–253, 1995.
7 T. M. Chan. Comparison-based time-space lower bounds for selection. In *SODA*, pages 140–149, 2009.
8 D. J. Elzinga and D. W. Hearn. Geometrical solutions for some minimax location problems. *Transportation Science*, 6(4):379–394, 1972.
9 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.
10 N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983.
11 N. Megiddo. The weighted euclidean 1-center problem. *Mathematics of Operations Research*, 8(4):498–504, 1983.
12 N. Megiddo and E. Zemel. An o(n log n) randomizing algorithm for the weighted euclidean 1-center problem. *J. Algorithms*, 7(3):358–368, 1986.
13 J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theor. Comput. Sci.*, 165(2):311–323, 1996.
14 M. I. Shamos and D. Hoey. Closest-point problems. In *FOCS*, pages 151–162, 1975.
15 J. J. Sylvester. A question in the geometry of situation. *Quarterly Journal of Mathemaitcs*, 1:79, 1857.
16 E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *Results and New Trends in Computer Science*, pages 359–370. Springer-Verlag, 1991.

Static Analysis for Checking Data Format Compatibility of Programs

Pranavadatta Devaki[1] and Aditya Kanade[2]

1 **IBM Research India**
2 **Indian Institute of Science**

Abstract

Large software systems are developed by composing multiple programs. If the programs manipulate and exchange complex data, such as network packets or files, it is essential to establish that they follow compatible data formats. Most of the complexity of data formats is associated with the headers. In this paper, we address compatibility of programs operating over headers of network packets, files, images, etc. As format specifications are rarely available, we infer the format associated with headers by a program as a set of guarded layouts. In terms of these formats, we define and check compatibility of (a) producer-consumer programs and (b) different versions of producer (or consumer) programs. A compatible producer-consumer pair is free of type mismatches and logical incompatibilities such as the consumer rejecting valid outputs generated by the producer. A backward compatible producer (resp. consumer) is guaranteed to be compatible with consumers (resp. producers) that were compatible with its older version. With our prototype tool, we identified 5 known bugs and 1 potential bug in (a) sender-receiver modules of Linux network drivers of 3 vendors and (b) different versions of a TIFF image library.

1998 ACM Subject Classification D.2.4 Software/Program Verification; D.2.12 Interoperability

Keywords and phrases Data format compatibility, producer-consumer/version-related programs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.522

1 Introduction

Large software systems are developed by composing multiple programs. For the composed system to work as desired, it is essential that the component programs are compatible with each other. Compatibility being a fundamental requirement in system integration, various techniques for analyzing compatibility have been proposed. Finite state machines have been used for checking compatibility of programs with respect to temporal aspects of method calls or action sequences [14], message sequences [18, 33], and typed accesses to sequential data [15]. When a component program evolves, the two versions can be compared to extend compatibility guarantees of the older version to the newer version. Incompatibilities in component upgrades have been identified by checking implication between logical summaries of observed behaviors of the two versions [26]. Automata learning and model checking have been combined for determining substitutability of programs [32, 11].

In many software systems, the component programs manipulate and exchange complex data like network packets or files. Checking compatibility of such programs involves comparing the data formats followed by them. A data format describes types and constraints on values of the data elements. To be *type-compatible*, the programs must use compatible types while accessing the same data elements. Type incompatibility between programs is observed frequently in practice. As an example, consider the bug report [1] which describes a type incompatibility between Atheros sender-receiver pair. For MAC frames that carry payload,

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 522–533

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

Figure 1 Mis-interpretation of a MAC ACK frame by an Atheros receiver: (a) Layout of the incoming ACK frame and (b) The type-incompatible layout computed by the receiver

the data following the control data is 4-byte aligned. ACK frames do not carry payload and their control data is followed immediately by a CRC checksum. Figure 1(a) shows an ACK frame received by the receiver. The receiver incorrectly computes the starting index of CRC for the ACK frame as 12 by computing the next 4-byte aligned index after 10 (end-index of the control data). As shown in Figure 1(b), it reads bytes $12-15$ as an integer containing the CRC. This access overlaps with the last two bytes of the integer stored at offsets $10-13$ and offsets $14-15$ of the remaining data of the incoming frame.

Another reason for incompatibilities is the complexity of constraints over data values. The formats of network packets, files, and images support a number of variants. For example, an image can be an RGB or a grayscale image. A MAC frame can be a data, control, or management frame. To distinguish between the variants, the format prescribes that the *producer* program should constrain specific elements of the data to pre-determined values. For example, a MAC frame is marked as a control frame by setting bits $2-3$ of the frame control field to binary 10. A *consumer* program should check the same elements for equivalent constraints to identify the variants. Thus, in addition to types, the compatibility between a producer-consumer pair also depends on whether *the consumer accepts all variants or only some variants* of the data generated by the producer. The latter case can give rise to compatibility errors. As described in the bug report [2], an Intel receiver treats frames of length < 30 as undersized (corrupt) and drops them. However, the minimum frame length depends on the variant the frame belongs to. The check against the hard-coded value of 30 results in frames belonging to a variant being dropped. As a result, repeated attempts at communication with a sender, that produces them, end in timeout of an ssh session.

The constraints required to identify variants of data are typically encoded in a pre-defined region of data, called the meta-data or *header*, *e.g.*, packet and file headers. We call the rest of the data as the contents. A header determines interpretation of the contents. The mismatch in processing of headers is therefore a key reason for data format incompatibilities (*e.g.*, see bug reports [1, 2, 3, 4, 5]). As the bug reports indicate, if incompatibilities are not detected during system integration, they can cause a variety of runtime errors like incorrect output [4], rejection of input [5], data corruption [1, 3], and non-termination [2]. In this paper, we analyze compatibility of programs operating over headers of network packets, files, images, etc. The headers are usually of fixed-length whereas the contents can be unbounded. We consider headers of fixed-length. Variable-length headers are also common but their lengths commonly vary over a finite set and are thus a simple extension of fixed-length headers.

As the data format specifications are not available to us, we design a static analysis of sequential C programs to infer them. The approach presented in this paper applies to any *fixed-length data* (including but not limited to headers) that programs may exchange, *e.g.*, serialized data structures. We shall simply refer to these as (fixed-length) data and their formats as data formats. A *data format* is formalized as a finite set of pairs of the form (g, ℓ), called *guarded layouts*, where g is a symbolic constraint over data elements, called the *guard*, and the *layout map* ℓ maps types to sets of offsets into the fixed-length data. The guard represents a set of data values and the layout map gives their type. We formalize compatibility relations in terms of guarded layouts for (a) producer-consumer programs

(b) two versions of producer (or consumer) programs. A compatible producer-consumer pair is free of type incompatibilities and mismatch in processing of variants of data. The version-related programs are executed individually without any direct data exchange between them. Nevertheless, they are expected to work with the same programs. A backward compatible producer (resp. consumer) is guaranteed to be compatible with consumers (resp. producers) that were compatible with its older version.

We are not aware of any approach for compatibility checking that models both types and constraints in data format specifications. The approaches [18, 33, 15] model types of data elements but *not* constraints over them. As noted earlier, constraints are essential for accurate description of formats and to find subtle compatibility bugs. The data description languages PacketTypes [27], DataScript [8], and PADS [17] do permit modeling of both types and constraints. LearnPADS++ [35] even generates data descriptions automatically from the example data. However, their goal is to generate parsers, type checkers, etc. for data formats. In contrast, our goal is to infer the formats from C programs and use them to check program compatibility. Further, the above compatibility checking approaches consider programs which access data *sequentially* from FIFO channels [18, 33] or as data streams [15]. In our case, a program accesses the data as an in-memory data structure. These accesses are *not* necessarily sequential. Unlike the other approaches, we formalize compatibility of *version-related* producer/consumer programs as well.

We have applied our prototype tool to check data format compatibility of several programs. The producer-consumer case studies include sender-receiver modules for IEEE 802.11 MAC frame headers of Linux network drivers of 3 vendors, namely, Atheros, Intel, and Intersil obtained from the Linux distributions. The version-related case study involves 48 functions operating over meta-data of TIFF files from 2 pairs of different versions of libtiff [6], a popular image processing library. We identify 5 known bug and 1 potential bug in these programs. In two cases, bug fixes are available for known bugs. Our static analysis confirms that the fixes resolve the compatibility issues. In summary,

- The paper defines data formats as guarded layouts for fixed-length data like packet and file headers. The formats are inferred by static analysis of C programs (Section 3).
- Compatibility relations over data formats and runtime guarantees provided by them are formalized for producer-consumer and version-related programs (Section 4).
- The approach is implemented and used for finding bugs and proving compatibility of real-world programs (Section 5). Finally, the work is compared with related approaches (Section 6) and future work is discussed (Section 7).

2 Overview

In this section, we present an overview of our approach with examples. Consider the programs producer and consumer given in Figure 2. These are inspired by sender and receiver modules that prepare and interpret network packet headers. We shall consider a modified version of producer to illustrate compatibility of version-related programs.

The program producer prepends a header to the frame received through buf. The user designates buf as its output variable. It can encode three variants of headers declared as struct hdr, qhdr, and lhdr. It accesses buf by overlaying buf with all of them. The program consumer reads the header from the (user-designated) input variable buf. For each of the designated variables, the user provides the starting offset and length of the sequential data that can be accessed through it. In the case of pointer variables, the accesses are by dereferencing the pointers. In this example, buf points to offset 0 in each program. The

```
1  int devmode, qosmode; // global variables configured by other components
2  struct hdr { short tods,fromds,type; char restofhdr[14]; int len; };
3  struct qhdr { short tods,fromds,type,qos; char restofhdr[14]; int len; };
4  struct lhdr { short tods,fromds,type; char restofhdr[20]; int len; };
```

```
5  int producer(char* buf, short qos, int blen) {
6    struct hdr* hd = (struct hdr*) buf;
7    struct qhdr* qhd = (struct qhdr*) buf;
8    struct lhdr* lhd = (struct lhdr*) buf;
9
10   if(devmode == 0) {
11     hd->tods = 0; hd->fromds = 1;
12     if(qosmode == 1) {
13       qhd->type = 1;
14       qhd->qos = qos;
15       qhd->len = blen + 26;
16     } else {
17       hd->type = 0;
18       hd->len = blen + 24;
19     }
20   } else if(devmode == 1) {
21     lhd->tods = lhd->fromds = 1;
22     lhd->type = 0;
23     lhd->len = blen + 30;
24   } else return ERR; // error return
25
26   return 0; // normal return
27 }
```

```
28 int consumer(char* buf) {
29   int len = 0;
30   struct hdr* hd = (struct hdr*) buf;
31
32   if(hd->len < sizeof(struct ldr))
33     return ERR; // error return
34
35   if(hd->tods == 1 && hd->fromds == 1)
36     return ERR; // error return
37
38   if(hd->tods == 0 && hd->fromds == 1)
39     len = hd->len;
40
41   return len; // normal return
42 }
```

Figure 2 Example of a producer-consumer pair of programs

other interface variables and global variables of producer are initialized by its environment and are not part of the data being exchanged. The length of the data is given to be 30. It can accommodate any of the three variants of the header. The user classifies the return statements into error and normal return as annotated in Figure 2.

The two programs do not run on the same machine. The actual transportation of data between them is handled by I/O routines which are not part of the analysis. Since the data exchange may happen across platforms, compatibility issues like differences in endian-ness may arise at lower-levels of abstraction. These issues are outside the scope of this paper.

Static inference of data formats. The two programs being checked for compatibility can use different type declarations and variable names, *e.g.*, programs developed by different vendors. Using structure declarations as data formats to check compatibility is not sufficient as they do not capture the constraints over data elements. Moreover, since data is often manipulated using pointers and type casts, structure declarations do not always indicate the layout map used to access the data. Hence, we infer data formats using static analysis. We represent data formats in a program-independent manner so that they can be compared. We consider the *primitive types* like characters and integers instead of user-defined types like hdr and qhdr. Thus, our layout maps are at a lower-level of abstraction than the types appearing in the programs. Further, we follow a *uniform naming convention of data elements* based on their offsets into the data and sizes. The guards are defined over these uniformly-named data elements. Since we want guarded layouts of output of a producer, we perform a forward analysis. Analogously, we perform a backward analysis of consumers.

In Figure 2, lines 6-8 type-cast and create aliases to buf. There are 3 paths in producer that reach the normal return at line 26. The return at line 24 is an error return and the output produced when this return is reached is not considered to be a valid output. Consider the path P through the first 2 if-branches (lines 10 and 12) and ending at the normal return.

- The data format (set of guarded layouts) of producer is $\{(g_1, \ell_1), (g_2, \ell_2), (g_3, \ell_3)\}$ where
  $g_1 : d[0:1] = 0 \wedge d[2:3] = 1 \wedge d[4:5] = 1$
  $\ell_1 : [short \mapsto \{0, 2, 4, 6\}, int \mapsto \{22\}, x \mapsto \{8, \ldots, 21, 26, \ldots, 29\}]$
  $g_2 : d[0:1] = 0 \wedge d[2:3] = 1 \wedge d[4:5] = 0$
  $\ell_2 : [short \mapsto \{0, 2, 4\}, int \mapsto \{20\}, x \mapsto \{6, \ldots, 19, 24, \ldots, 29\}]$
  $g_3 : d[0:1] = 1 \wedge d[2:3] = 1 \wedge d[4:5] = 0$
  $\ell_3 : [short \mapsto \{0, 2, 4\}, int \mapsto \{26\}, x \mapsto \{6, \ldots, 25\}]$
- The data format of consumer is $\{(g_4, \ell_4)\}$ where
  $g_4 : d[0:1] = 0 \wedge d[2:3] = 1 \wedge d[20:23] \geq 30$
  $\ell_4 : [short \mapsto \{0, 2\}, int \mapsto \{20\}, x \mapsto \{4, \ldots, 19, 24, \ldots, 29\}]$

Figure 4 Data formats inferred from producer and consumer programs of Figure 2

The set of typed-accesses to the data through the pointers along this path (see Figure 3) determine the layout map. The layout map according to these accesses is given as ℓ_1 in Figure 4. The type x is a special type, called *don't care*, which indicates that there was no access to the byte. It allows us to detect if two programs make non-identical accesses.

Now, consider the symbolic path condition and program state at the end of the path. As the variables devmode, qosmode, qos, and blen are not among the designated variables, we forget constraints over them. This is because these variables are not visible to the consumer and its processing of the data cannot depend on them. In order to compare guards across programs, we uniformly name the fields accessed through hd, qhd, and lhd as $d[i : j]$ where i is the starting offset of the field and $j = i + k - 1$ with k equal to size of the field. The guard g_1 corresponding to the layout map ℓ_1 is shown in Figure 4. As the values of arguments qos and blen are unconstrained, the values of data elements $d[6 : 7]$ and $d[22 : 25]$ assigned from them also remain unconstrained and are not shown in g_1. For consumer, there is only one path that leads to a normal return (line 41). The data formats of the programs are given in Figure 4.

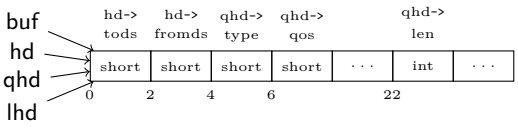


Figure 3 Pictorial view of the typed-accesses along path P to the data prepared by producer

Producer-consumer compatibility. The compatibility of a pair of programs can be checked using the inferred data formats. One of the program acts as a producer and another as a consumer. The same program may act as a producer or a consumer depending on the context. Programs like network protocol implementations exchange data interactively and the complete analysis of compatibility between them additionally requires analysis of temporal ordering of data exchanges. Our computation of data formats yields the set of guarded layouts exchanged from all possible protocol-states (if any) of such a program. This suffices to identify *data format* incompatibilities.

As a compatibility question, we can ask (1) whether the consumer accesses data elements with types compatible with those of the producer? The answer for our example is no, as $g_1 \wedge g_4$ is satisfiable but $\ell_4(int) \not\subseteq \ell_1(int)$. As the conjunction of guards is satisfiable, there is some datum generated by producer that consumer accepts. However, consumer accesses offsets 20-23 of the datum as an *int* unlike producer. We can also ask (2) whether consumer accepts *every* output of producer (in a type-compatible manner)? The answer is no, as $g_2 \not\Rightarrow g_4$, since $d[20 : 23]$ (denoting hd−>len) is unconstrained in g_2. We shall formalize the above questions

respectively as *weak* and *strong* compatibility requirements.

Compatibility of version-related programs. Let us now consider a newer version of producer by removing lines 20-23. Can we substitute the newer version in place of the older version? The answer in this case is affirmative. The data format of the newer version is $\{(g_1, \ell_1), (g_2, \ell_2)\}$. Thus, the newer version produces a subset of the outputs produced by the older version and associates the same layout maps to the common outputs. We call the newer version of the producer *backward* compatible in this case. If a consumer is type-safe (weakly compatible) with the older version of producer, then it will be type-safe with the newer version also. We shall similarly infer how strong compatibility is preserved by a backward compatible producer. Backward compatibility is also defined for consumers.

False positives and negatives. The guards inferred by a sound static analysis can be over-approximations of conditional accesses to the data by the programs. The over-approximations in points-to information can also cause imprecision. A compatibility result with imprecise formats can be a false positive or negative (discussed in Section 4). Similar limitations apply to related approaches [26, 15]. In the case of [26], the limitation stems from implication checks between dynamically inferred constraints which can be unsound (under-approximate) or imprecise (over-approximate). The automata computed in [15] are over-approximate. Therefore, the language containment between such automata may result in a false positive or negative. We control the false positives and negatives for our case studies by designing a sufficiently precise analysis with a suitable choice of abstract domain.

3 Static Inference of Data Formats

Since C programs routinely use pointers, we perform a points-to analysis to identify (abstract) memory locations that a pointer may point to. The precision of points-to information is important to us to avoid coarse over-approximations of the analysis results. We therefore perform a flow-sensitive may points-to analysis based on [16]. Points-to analysis and the memory model used are discussed further in the extended version [7]. We first formalize the notion of data formats for fixed-length data.

▶ **Definition 1.** A *data format* is a finite non-empty set of pairs of the form (g, ℓ), called *guarded layouts*, where g is a symbolic constraint over data elements, called the *guard*, and the *layout map* ℓ maps primitive types to subsets of the set OL of offset-based locations.

Intuitively, every guarded layout represents a variant supported by the data format and encoded in the program. For a layout map ℓ and a type t, if $k \in \ell(t)$ then it indicates that a *data element* of type t resides at offset k and it occupies offsets $\{k, \ldots, k + sizeof(t) - 1\}$. Every offset-based location in OL must belong to the set of offsets occupied by exactly one data element. The type of an offset is inferred only if it is accessed in the program. Unaccessed bytes are assigned the *don't care* type x. As discussed in Section 2, if an element of type t resides at offset i then it is named as $d[i : j]$ where $j = i + sizeof(t) - 1$. The guards are symbolic constraints over these uniformly-named data elements.

In order to compute a sound over-approximation of the set of guarded layouts, we perform abstract interpretation [13]. Our analysis computes layout maps and the corresponding sets of reachable states at every control location in the program. The reachable state-sets are represented using an abstract domain. The abstract domain is defined over uniformly-named data elements as well as program variables which do not refer to the offset-based locations. This is essential to capture all possible dependences between program variables. We call these as *extended guards*. The data format (the set of guarded layouts) is obtained by restricting

the extended guards to the data elements, at appropriate control locations. For a producer, the format is obtained at program exits whereas for a consumer it is obtained at the program entry. Due to space constraints, we discuss analysis of producers only.

Abstract domains. Let $\mathcal{G} = (G, \sqsubseteq_G, \sqcup_G, \sqcap_G, \bot_G, \top_G)$ be an abstract lattice to represent extended guards, with $\sqsubseteq_G$ as the partial order, $\sqcup_G$ as join, $\sqcap_G$ as meet, and $\bot_G$, $\top_G$ resp. as bottom and top elements. Since the values of data elements can range over infinite domains, we require $\mathcal{G}$ to be equipped with a widening operator. Let $\mathcal{L} = (L, \sqsubseteq_L, \sqcup_L, \sqcap_L, \bot_L, \top_L)$ be the *finite* lattice of layout maps of a fixed length. We have $\ell \sqsubseteq_L \ell'$ if $\ell(x) \supseteq \ell'(x)$ and for each primitive type $t \neq x$, $\ell(t) \subseteq \ell'(t)$. Let $D = \mathcal{G} \times \mathcal{L}$ be the domain of layout maps with extended guards. We denote an extended guard with $\hat{g}$ and a guard restricted to only data elements by g.

Transfer functions. Let $(\hat{g}, \ell) \in D$ be the abstract element before the current control location. If the statement contains a pointer r that may point to multiple memory locations then, for each of the memory locations, we create a *separate* transformed guarded layout. Thus, $(\hat{g}, \ell)$ may map to a *set* of guarded layouts. This is to keep the guarded layouts precise individually at the cost of increase in the number of guarded layouts. The guarded layout $(\hat{g}, \ell)$ is updated by considering the *same memory location* simultaneously in transfer functions of *both* $\hat{g}$ and ℓ. Within a loop, we consider all memory locations simultaneously. The transfer functions for extended guards are defined by the chosen abstract interpretation over the domain $\mathcal{G}$. The choice of $\mathcal{G}$ is an implementation issue.

The transfer functions for layout maps are now presented. The types of offset-based locations are inferred based on accesses to them. For a producer, we infer the layout maps by considering both read and write accesses to offset-based locations. The offset-based locations can be accessed through any valid C expression involving a user-designated variable or pointer (and aliases to it). The space of syntactic expressions is large and we consider only some representative cases. Let us denote a pointer by r, a structure (or union) by s, a structure field by f, a type by t, and a scalar variable by v. Let us denote the points-to map at a control location by PM and the compiler assigned type of an expression e by $\mathsf{typeof}(e)$. Consider the commonly used expressions v, $*r$, $s.f$, and $r\texttt{->}f$. Let E be an expression occurring at the current control location and $\mathsf{subexp}(E)$ be the set of subexpressions of E which fall into the above expression classes. An incoming layout map ℓ is potentially updated by each subexpression $e \in \mathsf{subexp}(E)$. Operators in E do not play any role in updating of layout maps. In the case of v and $s.f$, an incoming layout map is updated only if v and s are user-designated variables. In the case of $*r$ and $r\texttt{->}f$, we check whether $PM(r) \cap OL$ is empty or not. If it is empty then the expression is ignored.

For a suitable expression e, the offset-based location k accessed by it is computed. For v, it is the user-designated offset. For $*r$, it is an offset-based location in $PM(r)$. For $s.f$ and $r\texttt{->}f$, let b be the user-given offset of (the first field of) s or an offset-based location in $PM(r)$. Suppose the type of s is $\texttt{struct T}$ and of r is pointer to $\texttt{struct T}$. The offset-based location k corresponding to $s.f$ or $r\texttt{->}f$ is obtained by incrementing b by the cumulative sum of sizes of fields preceding the field f in $\texttt{struct T}$. The type of k in the layout map ℓ is changed to $t = \mathsf{typeof}(e)$ as follows: $\ell' := \ell[t \mapsto \ell(t) \cup \{k\}, x \mapsto \ell(x) \setminus \{k, \ldots, k + sizeof(t) - 1\}]$. If an offset ends up getting assigned to multiple data elements in a layout map, then we have found a potential *type error* and the analysis stops.

For example, the incoming set of guarded layouts at Line 11 of Figure 2 consists of a single guarded layout $(\hat{g}, \ell)$ where $\hat{g}$ is $\texttt{devmode} = \texttt{0}$ (corresponding to the predicate of the conditional at Line 10) and ℓ is $\bot_L$. Note that $\hat{g}$ is an extended guard and hence is not

restricted to constraints over data elements only. ℓ is $\bot_L$ as no access to data is made yet in the program. At Line 11, $(\hat{g}, \ell)$ is transformed into $(\hat{g}', \ell')$, where $\hat{g}'$ is devmode $= 0 \wedge d[0:1] = 0 \wedge d[2:3] = 1$ and ℓ' is $[short \mapsto \{0, 2\}, int \mapsto \{\}, x \mapsto \{4, \ldots, 29\}]$, to capture the effect of the assignment statements and the accesses to data at offsets 0 and 2 (through hd−>tods and hd−>fromds).

The overall analysis. Based on our experience with the case studies, we realize that maintaining branch-correlations is useful. We therefore perform a forward path-sensitive analysis by taking union of sets of guarded layouts at join points (outside loops). For a loop, the incoming set of guarded layouts is treated as an ordered set. The termination of the analysis is ensured by avoiding (potentially unbounded) growth in the number of guarded layouts and using the widening operator of domain $\mathcal{G}$. For statements within a loop, the transfer functions map a guarded layout to a single transformed guarded layout and at join points, we take element-wise join of the ordered sets of guarded layouts.

We compute procedure summaries bottom-up and use them at procedure call sites. Let $(\hat{g}, \ell)$ be an extended guarded layout at a call site. The constraints over variables or memory locations in the calling context that are written into by the callee are forgotten from $\hat{g}$. If a formal parameter is not written into by the callee then its constraints in the procedure summary are transferred to the corresponding actual. The guard $\hat{g}$ is combined with every guard $\hat{g'}$ in the procedure summary as $\hat{g} \sqcap_G \hat{g'}$. The matching layout map is obtained by $\ell \sqcup_L \ell'$. Finally, the assignment of the return value to the LHS of the call (if any) is processed. The guard is then restricted to variables in the calling context.

4 Compatibility Relations

We assume that precise data format specifications, in the form of sets of guarded layouts, are available. We define the compatibility relations over them and discuss the effect of *imprecision* in the format specifications, obtained by static analysis, subsequently.

Consider two layout maps ℓ and ℓ' defined over the same set of offset-based locations. The layout map ℓ is *compatible* with ℓ', denoted by $\ell \preceq \ell'$, if $\ell(x) \supseteq \ell'(x)$ and for every $t \neq x$, $\ell(t) \subseteq \ell'(t)$. Equality of layout maps $(=)$ is obtained by replacing $\subseteq$ and $\supseteq$ with $=$ in the above definition. If $\ell \preceq \ell'$, then every datum produced with layout map ℓ' can be consumed at runtime with layout map ℓ without type incompatibilities. Given a set G of guarded layouts of a program, the program is *locally compatible* if the guards in G with different layout maps are pairwise disjoint. If a datum can be produced with different layout maps, consumers cannot know which layout map to use for it. In the following discussion, we assume that the programs are locally compatible.

Producer-consumer programs. We define two compatibility relations between producer-consumer programs. Weak compatibility guarantees that if a consumer consumes a datum produced by a producer, it accesses it using a compatible layout map. Strong compatibility ensures, in addition, that every datum produced by a producer is consumed by the consumer. Strong compatibility implies weak compatibility.

▶ **Definition 2.** Let G and G' be sets of guarded layouts representing the data formats of a producer program P and a consumer program C respectively. P is *weakly compatible* with C, denoted by $WC_{\preceq}(P, C)$, if for every $(g, \ell) \in G$ and $(g', \ell') \in G'$, if $g \wedge g'$ is satisfiable, then $\ell' \preceq \ell$. P is *strongly compatible* with C, denoted by $SC_{\preceq}(P, C)$, if for every $(g, \ell) \in G$, there exists a $(g', \ell') \in G'$ such that $g \Rightarrow g'$ and $\ell' \preceq \ell$.

Stricter variants of the above definitions, $WC_{=}(P, C)$ and $SC_{=}(P, C)$, are obtained by replacing layout compatibility ($\preceq$) with layout equality ($=$) in them. These are useful to identify whether the two programs access the same set of offsets or not.

Version-related programs. Our definition of backward compatibility allows a newer version of a producer to initialize more offsets of the data, but restricts it to produce a subset of values taken by the data elements. If the producer initializes less offsets then they remain unconstrained and the set of data values produced is more. For a newer version of a consumer, consuming more data values is allowed, but reading from more offsets of the data is disallowed.

▶ **Definition 3.** Let G and G' be sets of guarded layouts representing the data formats of two versions P and P' of a producer or two versions C and C' of a consumer.

1. The producer P' is *backward compatible* with P, denoted by $BC_{\preceq}(P', P)$, if for every $(g', \ell') \in G'$, there exists $(g, \ell) \in G$ such that $g' \Rightarrow g$ and $\ell \preceq \ell'$.
2. The consumer C' is *backward compatible* with C, denoted by $BC_{\preceq}(C', C)$, if for every $(g, \ell) \in G$, there exists $(g', \ell') \in G'$ such that $g \Rightarrow g'$ and $\ell' \preceq \ell$.

Stricter versions of backward compatibility can be obtained by using layout equality ($=$) in place of layout compatibility ($\preceq$). A backward compatible producer can be substituted for its older version while preserving the compatibility relations of the older version. A backward compatible consumer preserves strong compatibility but does *not* necessarily preserve weak compatibility. The precise formulation of compatibility relations preserved by backward compatible programs and their proofs are discussed in the extended version [7].

Analysis of false positives and negatives. The guards obtained by static analysis can be over-approximations of the precise constraints associated with a layout map. Both local and weak compatibility definitions require checking satisfiability of $g \wedge g'$. Since the check is done using the over-approximated guards, the satisfiability can be spuriously true. Thus, compatibility of layout maps of disjoint guards may have to be checked and it can fail. Hence, local and weak compatibility checks are sound with respect to guards, but can result in false positives. The layout maps can also over-approximate the sets of typed-accesses. Thus, compatibility check over them may result in a false positive or negative, due to set inclusion over offset-sets used in layout map compatibility. The strong and backward compatibility require checking implication between guards. Because of over-approximation, this check can result in either a false positive or negative.

5 Experimental Evaluation

This section summarizes the experimental results. We have prototyped our approach in OCaml using CIL [29] and the octagon library [28]. The complexity of most of the operations over octagons is cubic in the size of the octagon.

Producer-consumer programs. We analyze sender-receiver modules of Linux (version 2.6.33.3) wireless LAN drivers of three vendors, namely, Atheros, Intel, and Intersil. They operate over IEEE 802.11 MAC frame headers. We bit-blast some fields of the headers. In addition, we consider a pair of programs from libtiff [6]: set/get routines that allow clients to get and set TIFF directory attributes. In all, 9 programs – 3 producers and 6 consumers – were analyzed. Of these, two consumers were with fixed versions of programs. These programs are a few hundreds of lines in size and the number of guarded layouts range from a few tens to over 100. All programs were analyzed in about 75 seconds. All variants of weak

No.	Programs	Bug Description	Failed Check
1.	Atheros-Atheros	Consumer reads offsets not written by the producer [1]	$WC_{\preceq}$
2.	Intersil-Intel	Valid packets are discarded by the consumer [2]	$SC_{\preceq}$
3.	TIFF set-get	An offset is written as a short and read as an int [3]	$WC_{\preceq}$
4.	tiffwrite (7 programs)	New version of the producers start producing images with field_planarconfig = 0 [4]	$BC_{\preceq}$
5.	tiffread (14 programs)	New version of the consumers access more offsets	$BC_{\preceq}$
6.	tiffcp	New version of the consumer accesses more offsets [5]	$BC_{\preceq}$

Figure 5 Description of data format compatibility bugs found by our tool

and strong compatibility were checked for 6 producer-consumer pairs. Figure 5 describes the bugs found by our tool. Both Atheros and Intersil-Intel bugs were discussed in the Section 1. The TIFF set-get routines fail the weak compatibility check as they access an offset with different types. All compatibility checks were evaluated in about 26 seconds.

Version-related programs. We analyze the TIFF directory manipulation routines of two versions, 3.7.4 and 3.9.4, of libtiff [6] and two versions, 3.5 and 3.6, of the TIFF copy tool tiffcp. In all, 42 libtiff routines (26 producers from tiffwrite module and 16 consumers from tiffread module) and 6 tiffcp routines (both as producers and consumers) were analyzed in about 7 minutes. These versions are several hundreds of lines of code in size and generate tens to over 150 guarded layouts. The newer versions of the producer routines of tiffwrite module produce images with field_planarconfig = 0, which the older version did not produce. Some of the tiffread routines of the newer version access more offsets of the data than the older version. This bug was not reported earlier. Similarly, one tiffcp consumer accesses more offsets than its older version. All compatibility checks were evaluated in about 21 seconds.

False positives and false negatives. The backward compatibility checks for tiffread and tiffcp fail due to two reasons: (1) For some guard g of the older version of a consumer, there is no guard g' in the newer version such that $g \implies g'$. (2) The newer consumer reads more offsets. The second reason is a true positive (bug). The first reason is a false positive for these examples. The newer consumers have case splits on a particular value. The associated layout maps are equal and if we take union of the guards then g implies the union. However, as union in octagonal domain is not precise, we do not perform it by default. There were no false negatives in the case studies.

6 Related Work

Compatibility checking. In Section 1, we have compared our work with related work on compatibility of producer-consumer programs [18, 33, 15]. Our intuition that a backward-compatible producer should produce less and a backward-compatible consumer should consume more is similar to behavioral subtyping [24] over class hierarchies. The refinement of interface automata [14] and implications between pre/post conditions in [26] model similar constraints for version-related programs. Substitutability analysis [32, 11] allows the newer version to exhibit more behaviors provided it preserves system-level safety properties.

Representation dependence testing [20] evaluates *forward compatibility* of consumers by simulating a large class of producers by sequentially composing a specification program and its algorithmically derived inverse. Compatibility tests have been generated from behavioral models of components which include sequences of actions and constraints over data being

exchanged [25]. However, the models are synthesized from observed behaviors of programs and do not identify type incompatibilities. Fingerprint generation [9] generates inputs that take two implementations of a protocol to different states. They symbolically execute binaries and use decision procedures to generate inputs.

Data format specification and inference. Data description languages [27, 8, 17] permit more expressive specifications than our guarded layouts, *e.g.*, they provide type constructors for unbounded data. However, their goal is to generate supporting tools like parsers and type checkers for analysis of data according to the specifications. Polyglot [10] extracts protocol message formats by dynamic analysis of binaries. Static analysis has been used to extract output formats from stripped executables [23] and data models from programs [21].

Type inference. Physical type checking [12] computes the in-memory layout of C structures to prove type safety of field accesses in presence of pointer type-casts. Our layout maps are defined over an ordered set of offset-based memory locations and are associated with guards. Type inferencing approaches [30, 22] have been used for extracting data abstractions from Cobol programs. Our idea of inferring concrete types based on accesses rather than on type declarations is similar to the use of access patterns in a specific program for identification of aggregate structures [30]. Our offset-based uniform naming is similar to their atomization notion. Our analysis is however aimed at C programs and we associate guards with layout maps. There are several other type systems, like dependent types for imperative languages [34], liquid types [31], and predicated type hierarchy [19], which associate guards with types.

7 Conclusions and Future Work

We presented an approach to check data format compatibility of C programs. Our approach infers data formats by static analysis of programs and checks various notions of compatibility of producer-consumer and version-related programs. It was shown to be effective in finding bugs in real programs. We plan to extend the approach to unbounded-length data and explore its use in improving regression testing on data formats. The integration of guarded layouts with finite state machine specifications of protocols can help in checking richer notions of compatibility of communicating programs.

Acknowledgments. We thank Satish Chandra and Nishant Sinha for their valuable feedback on the earlier versions of this paper.

References

1 http://marc.info/?l=linux-wireless&m=122888546315200&w=2.
2 http://bugzilla.intellinuxwireless.org/show_bug.cgi?id=169.
3 http://bugzilla.maptools.org/show_bug.cgi?id=2175.
4 http://bugzilla.maptools.org/show_bug.cgi?id=2057.
5 http://www.asmail.be/msg0055485608.html.
6 http://www.libtiff.org/.
7 http://www.csa.iisc.ernet.in/~kanade/fsttcs-extended.pdf.
8 G. Back. Datascript - a specification and scripting language for binary data. In *GPCE*, pages 66–77, 2002.
9 D. Brumley, J. Caballero, Z. Liang, J. Newsome, and D. Song. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation. In *USENIX Security Symposium*, pages 15:1–15:16, 2007.

10 J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In *CCS*, pages 317–329, 2007.

11 S. Chaki, E. M. Clarke, N. Sharygina, and N. Sinha. Verification of evolving software via component substitutability analysis. *FMSD*, 32(3):235–266, 2008.

12 S. Chandra and T. Reps. Physical type checking for C. In *PASTE*, pages 66–75, 1999.

13 P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.

14 L. de Alfaro and T. Henzinger. Interface automata. In *FSE*, pages 109–120, 2001.

15 E. Driscoll, A. Burton, and T. Reps. Checking conformance of a producer and a consumer. In *FSE*, pages 113–123, 2011.

16 M. Emami, R. Ghiya, and L. Hendren. Context-sensitive interprocedural points-to analysis in the presence of function pointers. In *PLDI*, pages 242–256, 1994.

17 K. Fisher and D. Walker. The PADS project: an overview. In *ICDT*, pages 11–17, 2011.

18 M. Fähndrich, M. Aiken, C. Hawblitzel, O. Hodson, G. Hunt, J.R. Larus, and S. Levi. Language support for fast and reliable message-based communication in Singularity OS. In *EuroSys*, pages 177–190, 2006.

19 R. Jhala, R. Majumdar, and R. Xu. State of the union: Type inference via Craig interpolation. In *TACAS*, pages 553–567, 2007.

20 A. Kanade, R. Alur, S. Rajamani, and G. Ramalingam. Representation dependence testing using program inversion. In *FSE*, pages 277–286, 2010.

21 R. Komondoor and G. Ramalingam. Recovering data models via guarded dependences. In *WCRE*, pages 110–119, 2007.

22 R. Komondoor, G. Ramalingam, S. Chandra, and J. Field. Dependent types for program understanding. In *TACAS*, pages 157–173, 2005.

23 J. Lim, T. Reps, and B. Liblit. Extracting output formats from executables. In *WCRE*, pages 167–178, 2006.

24 B. Liskov and J. Wing. Behavioral subtyping using invariants and constraints. In *Formal Methods for Distributed Processing: An Object Oriented Approach*, pages 254–280. 2001.

25 L. Mariani, S. Papagiannakis, and M. Pezzè. Compatibility and regression testing of COTS-component-based software. In *ICSE*, pages 85–95, 2007.

26 S. McCamant and M.D. Ernst. Early identification of incompatibilities in multi-component upgrades. In *ECOOP*, pages 440–464, 2004.

27 P.J. McCann and S. Chandra. Packet types: Abstract specifications of network protocol messages. In *SIGCOMM*, pages 321–333, 2000.

28 A. Miné. The octagon abstract domain. *CoRR*, abs/cs/0703084, 2007.

29 G.C. Necula, S. McPeak, S.P. Rahul, and W. Weimer. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. In *CC*, pages 213–228, 2002.

30 G. Ramalingam, J. Field, and F. Tip. Aggregate structure identification and its application to program analysis. In *POPL*, pages 119–132, 1999.

31 P. M. Rondon, M. Kawaguchi, and R. Jhala. Low-level liquid types. In *POPL*, pages 131–144, 2010.

32 N. Sharygina, S. Chaki, E.M. Clarke, and N. Sinha. Dynamic component substitutability analysis. In *FM*, pages 512–528, 2005.

33 Z. Stengel and T. Bultan. Analyzing singularity channel contracts. In *ISSTA*, pages 13–24, 2009.

34 H. Xi. Imperative programming with dependent types. In *LICS*, pages 375–387, 2000.

35 K. Q. Zhu, K. Fisher, and D. Walker. Learnpads++: Incremental inference of ad hoc data formats. In *PADL*, pages 168–182, 2012.

The Complexity of Quantitative Information Flow in Recursive Programs

Rohit Chadha[1] **and Michael Ummels**[2]

1 **University of Missouri, Columbia**
`chadhar@missouri.edu`
2 **Technische Universität Dresden**
`ummels@tcs.inf.tu-dresden.de`

Abstract

Information-theoretic measures based upon mutual information can be employed to quantify the information that an *execution* of a program reveals about its *secret inputs*. The *information leakage bounding problem* asks whether the information leaked by a program does not exceed a given threshold. We consider this problem for two scenarios: a) the *outputs* of the program are revealed, and b) the *timing* (measured in the number of execution steps) of the program is revealed. For both scenarios, we establish complexity results in the context of deterministic boolean programs, both for programs with and without recursion. In particular, we prove that for recursive programs the information leakage bounding problem is no harder than checking reachability.

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.4.6 Security and Protection, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases quantitative information flow, recursive programs, program analysis, verification, computational complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.534

1 Introduction

Ensuring that a program preserves confidentiality of its *secret inputs* is a fundamental problem in security. Typically, one desires that the execution of the program reveals absolutely *no* information about its secret inputs. This desired property is often modeled as *non-interference* [18, 29] — the *low-security observations* of the execution of a program should be independent of the *high-security inputs*. These observations could be *explicit* outputs of the program (e.g., the results of an election or whether a password is correct or not), or they could be *implicitly* extracted from its execution (such as timing information, cache size or power consumption).

In practice, however, non-interference is hard to achieve as it often clashes with functionality. An unanimous election, for example, reveals the votes of each voter. Consequently, alternative approaches that aim to quantify the amount of information leakage have been proposed in the literature [15, 19, 25, 31]. In these information-theoretic approaches, programs are viewed as *transformers* of random variables — they transform a random variable taking values from the set of inputs into a random variable taking values from the set of observations. Intuitively, the amount of information leaked by the program is the difference between the initial uncertainty and the uncertainty remaining in the high-security inputs given the observations from running the program; the formal definition relies on the notion of *mutual information*, which is based on the seminal work of Shannon [30].

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 534–545

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

Given the importance of this problem, several automated approaches have been proposed to compute the information leaked by the program. The techniques employed by these approaches range from model-checking [3, 24, 10, 11] and static analysis [12, 13, 14, 3] to statistical analysis [24, 9]. From a more theoretical viewpoint, the *complexity* of computing the amount of leakage was only considered recently [34, 33, 35, 7]. More precisely, [33, 35, 7] consider the complexity of the *information leakage bounding problem*: given a program P, a distribution μ on the set of inputs, and a rational number q, check if the information leaked by the program (denoted by $\mathrm{SE}_\mu(P)$) does not exceed q.

In [33, 35], the program P is described in a simple non-recursive deterministic imperative language with boolean variables, assignments, conditionals and loops, and the inputs are assumed to be uniformly distributed. They show that the information leakage bounding problem is PP-hard for the loop-free fragment.[1] For the whole language with loops, the problem is shown to be PSPACE-hard. However, no upper bounds are given in [33, 35]. An EXPSPACE upper bound can be derived from the work of Černý et al. [7], where the information leakage bounding problem is shown to be PSPACE-complete under the assumption that the program is represented explicitly as a nondeterministic transition system and the input distribution is given explicitly. In our setting, an exponential blow-up occurs because the translation from a boolean program to a nondeterministic transition system is exponential.

Contributions. Our first contribution is an upper bound for loop-free boolean programs when the number of output variables is logarithmic in the size of the program.[2] We show that in this case the information leakage bounding problem for uniformly distributed inputs lies in the fourth level of the *counting hierarchy* (more precisely in $\mathsf{P}^{\mathsf{CH}_3}$). The whole counting hierarchy is contained in PSPACE. The main challenge in establishing the upper bound is that we have to solve inequations that involve logarithms (because of the definition of mutual information). In order to overcome this challenge, we resort to recent breakthroughs in arithmetic circuit complexity [2]. We then employ similar techniques to establish PSPACE-completeness for boolean programs with loops (but no recursion) under the same assumption that the number of output variables is logarithmic in the size of program. Hence, our upper bound is a substantial improvement over the previous EXPSPACE upper bound.

We subsequently turn our attention to boolean programs with recursion. We show that both the problem of checking non-interference as well as the information leakage bounding problem is EXPTIME-complete. For the upper bound, we observe that a recursive boolean program can be represented as an *exponential-size* deterministic pushdown system [26, 28] (the pushdown system is of size linear in the length of the program but exponential in the number of variables). We can then use the fact that control state reachability in pushdown systems is polynomial-time decidable [5] and thus compute the outputs of the program on any given input. A careful analysis of the expression computing information leakage then gives us the desired upper bound. We make no assumptions on the number of variables in this case, and hence this also gives an EXPTIME upper bound for general non-recursive programs, which is better than the EXPSPACE upper bound that can be derived from [7].

In the second part of this paper, we consider the case when the attacker can observe the timing behavior of an execution of a program. We abstract the timing behavior of the program by the "length" of the computation of the pushdown system corresponding to the program. One could alternatively use the number of procedure calls or the number

[1] Recall that PP is the class of decision problems decidable by a probabilistic polynomial-time Turing machine with acceptance probability $\geq 1/2$.

[2] If one also allows *low-level* security input variables, their number must also be bounded.

of loop executions as an abstraction of timing, but our results would not change in that case. For non-recursive terminating programs, the execution time can easily be measured "inside the program" by a binary counter, so bounding the information leaked by timing is no harder than the problem of bounding the information leaked by observing the outputs. The same idea does not work for recursive programs because the running time of a recursive program could be doubly exponential. Nevertheless, we show that the problem of bounding the information leaked by the timing behavior of a recursive boolean program on uniformly distributed inputs is also EXPTIME-complete by showing that the execution time for recursive terminating programs can still be computed in exponential time.

Related work. The complexity of quantitative information flow security for boolean programs was first tackled by Yasuoka and Terauchi [34], where the complexity of the information leakage comparison problem is studied: this problem asks which of two programs leaks more information. They also show that the problem of checking non-interference for loop-free programs is coNP-complete. The complexity of bounding information leakage was first studied by the same authors in [33], where the problem was shown to be PP-hard for loop-free non-recursive programs. In [35], the same authors prove that deciding non-interference for non-recursive programs *with loops* is PSPACE-complete. However, none of these papers contains an upper bound for the problem of bounding Shannon-entropy based information leakage, not even for restricted programs. Thus, our results obtained by restricting the number of output variables are novel. Only for the related notions of *min-entropy* and *guessing entropy*, a PSPACE upper and lower bound for non-recursive programs was established in [35].

A more general setting has been considered by van der Meyden and Zhang [32] as well as Černý et al. [7], where programs are represented abstractly as nondeterministic transition systems. In this setting, van der Meyden and Zhang established PSPACE-completeness for noninterference, and Černý et al. extended this result to the information-leakage bounding problem (wrt. Shannon entropy). However, as they assume an explicit-state description and the translation of a boolean program into an equivalent explicit-state description causes an exponential blowup, their results only give an EXPSPACE-upper bound for boolean programs (without recursion). None of these works consider recursive programs or the problem of bounding the information leakage caused by timing information. We establish EXPTIME-completeness for both problems, and also obtain better bounds for non-recursive programs.

Several timing attacks are known in literature. For example, [6] shows a practical timing attack against OpenSSL, which allows extraction of a private RSA key. The attack exploits the fact that the multiplication in OpenSSL is carried out by the Karatsuba routine [21], which is a recursive algorithm. Several approaches have been proposed in the literature to counteract timing leaks. Type systems, for example, are used to detect information leakage from timing [20], while [1, 23, 27, 4] provide countermeasures to combat information leakage from timing. None of these works have considered complexity questions, though.

Note. Due to space constraints, most proofs are only sketched or omitted entirely. For details, see the full version of this paper [8].

2 Preliminaries

All logarithms are to the base 2. As is standard, we define $0 \log 0 = 0$. We assume that the reader is familiar with probability distributions and (discrete) random variables. Given a function $f\colon A \to B$ and $b \in B$, the set $\{a \in A \mid f(a) = b\}$ is denoted by $f^{-1}(b)$. Finally, we denote by 2^A the set of all (total) functions from A to the set $\{\top, \bot\}$.

Straight-line programs and the counting hierarchy. A (division-free) *straight-line program* is a finite list of instructions of the form $x \leftarrow c$ or $x \leftarrow y \odot z$, where $c \in \{0,1\}$, $\odot \in \{+,-,\cdot\}$ and x, y, z are taken from a countable set of variables. Such a program is *closed* if all variables that appear on the right-hand side of an instruction also appear on the left-hand side of a preceding instruction. Hence, a closed straight-line program represents an integer, namely the value of the last variable that is assigned to. The problem PosSLP is to decide, given a closed straight-line program, whether the corresponding integer is > 0.

The counting hierarchy consists of the classes CH_i where $\mathsf{CH}_0 = \mathsf{P}$ and $\mathsf{CH}_{i+1} = \mathsf{P}^{\mathsf{CH}_i}$ for all $i \in \mathbb{N}$. Allender et al. [2] recently showed that the Problem PosSLP belongs to the complexity class $\mathsf{P}^{\mathsf{CH}_3}$ and thus to the fourth-level of the counting hierarchy. Since the counting hierarchy is contained in PSPACE, we know in particular that PosSLP is decidable in polynomial space.

Pushdown Systems. The operational semantics of recursive programs are given by pushdown systems. Formally a pushdown system (PDS) $\mathcal{P}$ is a tuple (Q, Γ, δ) where Q is a finite set of *control states*, Γ is a finite *stack alphabet*, and $\delta = \delta_{\text{int}} \cup \delta_{\text{cll}} \cup \delta_{\text{rtn}}$ is a finite set of *transitions* s.t. $\delta_{\text{int}} \subseteq Q \times Q$, $\delta_{\text{cll}} \subseteq Q \times Q \times \Gamma$, and $\delta_{\text{rtn}} \subseteq Q \times \Gamma \times Q$.

A labeled transition system $(\text{Labels}, \text{Conf}_{\mathcal{P}}, \to_{\mathcal{P}})$ defines the semantics of a PDS $\mathcal{P}$. The set Labels of labels is $\{\text{int}, \text{cll}, \text{rtn}\}$. The set $\text{Conf}_{\mathcal{P}}$ of configurations is $Q \times \Gamma^*$. The word $w \in \Gamma^*$ in a configuration (q, w) models the contents of the stack; the empty word ε denotes the empty stack. The transition relation $\to_{\mathcal{P}}$ is defined as follows: $(q, w) \xrightarrow{\text{int}}_{\mathcal{P}} (q', w)$ if $(q, q') \in \delta_{\text{int}}$; $(q, w) \xrightarrow{\text{cll}}_{\mathcal{P}} (q', wa)$ if $(q, q', a) \in \delta_{\text{cll}}$ and $(q, wa) \xrightarrow{\text{rtn}}_{\mathcal{P}} (q', w)$ if $(q, a, q') \in \delta_{\text{rtn}}$.

We omit the subscript $\mathcal{P}$ if it is clear from the context. Since we consider only deterministic programs, we are mainly interested in *deterministic* PDS: $\mathcal{P}$ is deterministic if for each s in $\text{Conf}_{\mathcal{P}}$ there is *at most one* $\lambda \in \text{Labels}$ and *at most one* $s' \in \text{Conf}_{\mathcal{P}}$ with $s \xrightarrow{\lambda}_{\mathcal{P}} s'$.

Given a configuration $c = (q, w)$ of a PDS $\mathcal{P}$, we say that $\text{state}(c) = q, \text{stack}(c) = w$ and $\text{height}(c) = |w|$, the length of w. A *computation* of $\mathcal{P}$ is a sequence $c_0 \xrightarrow{\lambda_1} \cdots \xrightarrow{\lambda_m} c_m$. A transition $c_i \xrightarrow{\text{cll}} c_{i+1}$ is a *procedure call*. Similarly, we define *procedure returns* and *internal actions*. We say that a procedure return $c_j \xrightarrow{\text{rtn}} c_{j+1}$ *matches* a procedure call $c_i \xrightarrow{\text{cll}} c_{i+1}$ iff $i < j$, $\text{height}(c_{i+1}) = \text{height}(c_j)$ and $\text{height}(c_{i+1}) \leq \text{height}(c_k)$ for all $i < k < j$. Finally, we say that $c \overset{m}{\Rightarrow}_{\mathcal{P}} c'$ if there exists a computation $c_0 \xrightarrow{\lambda_1} \cdots \xrightarrow{\lambda_m} c_m$ of $\mathcal{P}$ with $c_0 = c$ and $c_m = c'$, and we write $c \Rightarrow_{\mathcal{P}} c'$ if $c \overset{m}{\Rightarrow}_{\mathcal{P}} c'$ for some $m \in \mathbb{N}$. The following is proved in [5].

▶ **Theorem 1.** *There are polynomial-time algorithms that, given a PDS $\mathcal{P}$, output the set $\{(q, q') \mid (q, \varepsilon) \Rightarrow_{\mathcal{P}} (q', \varepsilon)\}$ and the set $\{(q, q') \mid \exists w \in \Gamma^* \ (q, \varepsilon) \Rightarrow_{\mathcal{P}} (q', w)\}$, respectively.*

Programs. Due to space constraints, we cannot present the syntax of recursive boolean programs in detail. Here we just highlight the main features. The inputs of our programs are partitioned into two sets, one containing *high-security variables* and one containing *low-security variables*. Additionally, our programs may have some local variables as well as outputs. The outputs are assumed to be of *low security*. Note that high-security outputs, i.e., outputs that are not visible to an observer, can easily be modeled using local variables.

We only give an informal description of the semantics of programs, which is *call-by-value*. A recursive boolean program can be represented as a deterministic pushdown system [26, 28] of exponential size (linear in the length of the program, but exponential in the number of variables). The states of the pushdown system keep track of the current statement and the values of all variables in the "current scope"; the pushdown stack keeps track of the procedure calls. Whenever a procedure is called, the pushdown system pushes the position of the call and the values of the variables onto the stack, transitions into the called procedure, and sets

all variables that are local to this procedure to $\perp$. Upon returning from the procedure call, the contents from the stack is popped and the variables are reset properly, i.e., the outputs of the returning procedure are set, and the variables that were local to the procedure are reset to their original values using the information from the stack. Since P is deterministic, the corresponding pushdown system is also deterministic.

The computation of the program P on high inputs $\bar{h}_0$ and low inputs $\bar{l}_0$ can now be defined as the computation of the pushdown system corresponding to P starting from the configuration with the empty stack and with the control state corresponding to the first statement of P, the input variables set to $\bar{h}_0, \bar{l}_0$, and the local and output variables set to $\perp$. The program P *terminates* on inputs $\bar{h}_0, \bar{l}_0$ if this computation reaches the configuration with the control state corresponding to the last statement of the program (in that case, the stack will be empty). If P terminates, we define the output of P to be the values of the output variables upon termination. Hence, P can be seen as a partial function $F_P \colon 2^{\bar{h}} \times 2^{\bar{l}} \to 2^{\bar{o}}$.

Henceforth, the program P is always assumed to be terminating. One could possibly model non-termination as an explicit observation; and our complexity results will not change in that case. This is because nontermination on an input can be decided for while programs in PSPACE and for recursive boolean programs in EXPTIME.

Quantifying information leakage. Let $\mathcal{X}$ be a discrete random variable with values taken from a finite set X. If μ is the probability distribution of $\mathcal{X}$, the *Shannon entropy* of μ, written $\mathrm{H}_\mu(\mathcal{X})$, is defined as

$$\mathrm{H}_\mu(\mathcal{X}) = -\sum_{x \in X} \mu(\mathcal{X} = x) \cdot \log \mu(\mathcal{X} = x).$$

If $\mathcal{X}$ and $\mathcal{Y}$ are discrete random variables taking values from finite sets X and Y with joint probability distribution μ, the *conditional entropy* of $\mathcal{X}$ given $\mathcal{Y}$, written $\mathrm{H}_\mu(\mathcal{X} \mid \mathcal{Y})$, is defined as

$$\mathrm{H}_\mu(\mathcal{X} \mid \mathcal{Y}) = \sum_{y \in Y} \mu(\mathcal{Y} = y) \cdot \mathrm{H}_\mu(\mathcal{X} \mid \mathcal{Y} = y),$$

where

$$\mathrm{H}_\mu(\mathcal{X} \mid \mathcal{Y} = y) = -\sum_{x \in X} \mu(\mathcal{X} = x \mid \mathcal{Y} = y) \cdot \log \mu(\mathcal{X} = x \mid \mathcal{Y} = y).$$

If $\mathcal{X}, \mathcal{Y}$ and $\mathcal{Z}$ are discrete random variables taking values from finite sets X, Y and Z with joint probability distribution μ, then the *joint conditional entropy* of $\mathcal{X}, \mathcal{Y}$ given $\mathcal{Z}$, written $\mathrm{H}_\mu(\mathcal{X}, \mathcal{Y} \mid \mathcal{Z})$ is the entropy of the random variable $(\mathcal{X}, \mathcal{Y})$ given $\mathcal{Z}$. Similarly, the conditional entropy of $\mathcal{X}$ given $\mathcal{Y}$ and $\mathcal{Z}$ is the entropy of $\mathcal{X}$ given $(\mathcal{Y}, \mathcal{Z})$.

If $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ are discrete random variables taking values from finite sets X, Y and Z respectively with joint probability distribution μ, then the *conditional mutual information* of $\mathcal{X}$ and $\mathcal{Y}$ given $\mathcal{Z}$, written $\mathrm{I}_\mu(\mathcal{X}; \mathcal{Y} \mid \mathcal{Z})$, is defined as

$$\mathrm{I}_\mu(\mathcal{X}; \mathcal{Y} \mid \mathcal{Z}) = \mathrm{H}_\mu(\mathcal{X} \mid \mathcal{Z}) - \mathrm{H}_\mu(\mathcal{X} \mid \mathcal{Y}, \mathcal{Z}).$$

We are interested in measuring the information leaked by a program. Following [15, 19, 25], we use conditional mutual information to quantify this information. As described above, we can view programs as functions that take two kinds of inputs: a *high-security* (high) input from a finite set H and a *low-security* (low) input from a finite set L. Let $\mathcal{H}$ and $\mathcal{L}$ be random variables taking values from H and L, respectively, with joint distribution μ.

Moreover, let O be a finite set and $F\colon H \times L \to O$ be a function. We extend μ to a joint probability distribution on $\mathcal{H}$, $\mathcal{L}$ and $\mathcal{O}$ such that

$$\mu(\mathcal{O} = o \mid \mathcal{H} = h, \mathcal{L} = l) = \begin{cases} 1 & \text{if } F(h,l) = o \\ 0 & \text{otherwise} \end{cases}$$

The *information leaked by the function* F is then

$$\mathrm{SE}_\mu(F) \coloneqq \mathrm{I}_\mu(\mathcal{H}; \mathcal{O} \mid \mathcal{L}).$$

We are mainly interested in the case where μ is the uniform distribution on $H \times L$, and we define $\mathrm{SE}_{\mathsf{U}}(F) \coloneqq \mathrm{SE}_\mu(F)$ in this case.

A function $F\colon H \times L \to O$ is *non-interferent* if $F(h,l) = F(h',l)$ for all $h, h' \in H$ and $l \in L$, and *interferent* otherwise. Note that a function $F\colon H \times L \to O$ is non-interferent iff $\mathrm{SE}_{\mathsf{U}}(F) = 0$ for all distributions μ.

Sometimes, we have only high inputs, i.e., F is a function from H to O. In that case, the information leaked by the function F is just $\mathrm{SE}_\mu(F) = \mathrm{I}_\mu(\mathcal{H}; \mathcal{O})$. The following lemma allows us to trade low inputs for high inputs and outputs.

▶ **Lemma 2.** *Let H, L, O be finite sets, $F\colon H \times L \to O$, and let $\mathcal{H}$ and $\mathcal{L}$ be random variables taking values in H and L with joint probability distribution μ. Consider the function $G\colon (H \times L) \to (O \times L)$ defined by $G(h,l) = (F(h,l), l)$. Then $\mathrm{SE}_\mu(G) = \mathrm{SE}_\mu(F) + \mathrm{H}_\mu(\mathcal{L})$.*

The following theorem is proved in [3, 22].

▶ **Theorem 3.** *Let H and O be finite sets, and let $F\colon H \to O$ be a function. Then*

$$\mathrm{SE}_{\mathsf{U}}(F) = \log|H| - \frac{1}{|H|} \sum_{o \in O} |F^{-1}(o)| \log |F^{-1}(o)|.$$

The information leakage bounding problem. As discussed above, a program P with high input variables $\bar{h}$, low input variables $\bar{l}$ and output variables $\bar{o}$ can be seen as a function $F_P\colon 2^{\bar{h}} \times 2^{\bar{l}} \to 2^{\bar{o}}$. Now, the information leakage bounding problem asks, given a program P and a rational number $q \geq 0$, whether the information leaked by F_P does not exceed q, i.e. whether $\mathrm{SE}_{\mathsf{U}}(F_P) \leq q$. In the rest of the paper, we will identify P with the function F_P.

3 Complexity of information leakage

Loop-free programs. We start by discussing our results for loop-free programs. Given numbers $a_1, \ldots, a_k \in \mathbb{N}$, we define $\sigma(a_1, \ldots, a_k) = \sum_{i=1}^{k} a_i \log a_i$. Note that if $F\colon H \to O$ is a function, and $a_1, \ldots, a_k$ is a permutation of $\{|F^{-1}(o)| \mid o \in O\}$, then $\mathrm{SE}_{\mathsf{U}}(F) = \log|H| - \sigma(a_1, \ldots, a_k)/|H|$, according to Theorem 3.

▶ **Lemma 4.** *Given $a_1, \ldots, a_k \in \mathbb{N}$ and $q \in \mathbb{Q}$, deciding whether $\sigma(a_1, \ldots, a_k) < q$ reduces to* PosSLP *in polynomial time.*

Proof. In order to prove the lemma, we show that, given $a_1, \ldots, a_k, q$, one can construct a (division-free) straight-line program S in polynomial time such that $\sigma(a_1, \ldots, a_k) < q$ iff $S \in \mathrm{PosSLP}$. Since $\sigma(a_1, \ldots, a_k)$ is always nonnegative, we can assume that $q > 0$. Let $q = r/s$, where both $r, s \in \mathbb{N} \setminus \{0\}$. Hence, $\sigma(a_1 \ldots, a_k) < q$ iff $s \cdot \sigma(a_1 \ldots, a_k) < r$. Using the fact that $\log a + \log b = \log ab$ and $a \log b = \log b^a$, we have

$$s \cdot \sigma(a_1, \ldots, a_k) = \log \prod_{i=1}^{k} a_i^{a_i s}.$$

Applying an exponentiation on both sides, we get that

$$\sigma(a_1 \ldots, a_k) < q \iff \prod_{i=1}^{k} a_i^{a_i s} < 2^r \iff 0 < 2^r - \prod_{i=1}^{k} a_i^{a_i s}.$$

Now, using repeated squaring, we can write a straight-line program of size $\mathrm{O}(\log r + \log s + \sum_{i=1}^{k} \log a_i)$ representing the number on the right-hand side, establishing our reduction. ◀

We can now show that the information leakage bounding problem for loop-free programs lies inside the counting hierarchy, provided the number of possible low inputs and outputs is only logarithmic in the number of high inputs.

▶ **Theorem 5.** *Given a loop-free program P with $|\bar{o}|+|\bar{l}| = \mathrm{O}(\log |\bar{h}|)$ and a rational number q, deciding whether $\mathrm{SE}_{\mathsf{U}}(P) \leq q$ can be done in $\mathsf{P}^{\mathsf{CH}_3}$.*

Proof. First, observe that if P has k low input variables and $\mathcal{L}$ is the distribution induced by U on low inputs, then $\mathrm{H}_{\mathsf{U}}(\mathcal{L}) = k$. Using this observation and Lemma 2, it suffices to consider the case when P has only high inputs. Moreover, since $\mathsf{P}^{\mathsf{CH}_3}$ is closed under complementation, it suffices to show that we can decide whether $\mathrm{SE}_{\mathsf{U}}(P) > q$ in $\mathsf{P}^{\mathsf{CH}_3}$.

Let $|\bar{h}| = m$ and denote by H and O the set of possible inputs and outputs, respectively. Note that $|H| = 2^m$ and $|O| = \mathrm{O}(m^d)$ for some $d \in \mathbb{N}$ (since $|\bar{o}| = \mathrm{O}(\log m)$). Let $O = \{\bar{o}_1, \ldots, \bar{o}_k\}$ and for each $i = 1, \ldots, k$ set $a_i = |P^{-1}(\bar{o}_i)|$. Now, by Theorem 3, we have

$$\mathrm{SE}_{\mathsf{U}}(P) = m - 2^{-m} \cdot \sigma(a_1, \ldots, a_k)$$

and therefore

$$\mathrm{SE}_{\mathsf{U}}(P) > q \iff \sigma(a_1, \ldots, a_k) < 2^m(m - q)$$

Note that all the numbers a_i as well as $2^m(m-q)$ are of size polynomial in the size of P and the size of q. Hence, given $a_1, \ldots, a_k$, we can apply Lemma 4 and compute (in polynomial time) a straight-line program S such that $S \in \mathrm{PosSLP}$ iff $\mathrm{SE}_{\mathsf{U}}(P) > q$.

Since PosSLP is in $\mathsf{P}^{\mathsf{CH}_3}$ [2], we are done if we can show that the numbers $a_1, \ldots, a_k$ can be computed by a polynomial-time algorithm with an oracle for CH_3. In fact, we show that these numbers can be computed in $\#\mathsf{P}$; since $\#\mathsf{P} \subseteq \mathsf{P}^{\mathsf{PP}}$, this will conclude the proof. Given an output $\bar{o}_i \in O$, the *weakest precondition semantics* gives us a Boolean formula $\varphi_i(\bar{h})$, which can be computed in polynomial time [17], such that an assignment $\alpha \in 2^{\bar{h}}$ satisfies φ_i iff $P(\alpha) = \bar{o}_i$. Hence, $a_i = |\{\alpha \mid \alpha \models \varphi_i\}|$. Since the problem of computing the number of satisfying assignments for a given Boolean formula is in $\#\mathsf{P}$, we are done. ◀

While programs. Non-interference for while programs is shown to be PSPACE-complete in [35]. Indeed, it can be shown to be PSPACE-hard even for programs that have only one high input variable, no low input variables, and one output variable. We show that the upper bound extends to the information leakage bounding problem, provided the number of possible low inputs and outputs is only logarithmic in the number of high inputs.

▶ **Theorem 6.** *Given a while program P with $|\bar{o}| + |\bar{l}| = \mathrm{O}(\log |\bar{h}|)$ and a rational number q, deciding whether $\mathrm{SE}_{\mathsf{U}}(P) \leq q$ is PSPACE-complete.*

Proof. We prove PSPACE hardness for the special case of one high input, no low inputs, and one output in [8]. The proof for containment in PSPACE is almost identical to the proof of Theorem 5. The only difference is that we cannot transform a while program into an

equivalent Boolean formula in polynomial time (or reachability for Boolean programs would be in NP). Instead, we just "run" the given program P on every possible input in order to compute the numbers $a_i = |P^{-1}(\bar{o}_i)|$, which can be done in polynomial space. Since the counting hierarchy is contained in PSPACE, this gives a polynomial-space algorithm. ◀

Recursive Programs. Deciding non-interference becomes EXPTIME-hard if we allow procedure calls, i.e., at least as hard as deciding reachability for recursive programs.

▶ **Theorem 7.** *Deciding non-interference for recursive programs with one high input, no low inputs, and one output is* EXPTIME*-hard.*

As a corollary, we get that the information leakage bounding problem for recursive programs is also EXPTIME-hard, even when the number of inputs and outputs is restricted. We now show that the information leakage bounding problem is indeed no harder than the reachability problem, i.e. is in EXPTIME. As opposed to our PSPACE upper bound for while programs, we will have no restriction on the number of inputs or on the number of outputs. In particular, the EXPTIME upper bound also applies to arbitrary while programs.

▶ **Theorem 8.** *The information leakage bounding problem is* EXPTIME*-complete for recursive programs.*

Proof. EXPTIME-hardness follows from Theorem 7. For the upper bound, as in the proof of Theorem 5, we can assume that P has m high inputs, no low inputs and n outputs, and that $0 \leq q < m$. Let H be the set of possible inputs to P and $O = \{\bar{o}_1, \ldots, \bar{o}_k\}$ the set of possible outputs. Hence, $|H| = 2^m$ and $k = |O| \leq 2^n$. As shown in the proof of Theorem 5, we have $\mathrm{SE}_{\mathsf{U}}(P) \leq q$ iff $\sigma(a_1, \ldots, a_k) \geq 2^m(m-q)$, where $a_i := |P^{-1}(\bar{o}_i)|$. Let $2^m(m-q) = r/s$, where $r, s \in \mathbb{N}$ (such numbers can be computed easily from P and q). Now, as in the proof of Lemma 4, we have

$$\mathrm{SE}_{\mathsf{U}}(P) \leq q \iff \log \prod_{i=1}^{k} a_i^{a_i s} \geq r.$$

Note that we have no restriction on the number of outputs. Hence, unlike in the proof of Theorem 5, we cannot appeal to Lemma 4. However, observe that $\sum_{i=1}^{k} a_i = 2^m$. Hence, by replacing the powers by products, we can write $p := \prod_{i=1}^{k} a_i^{a_i s}$ as a product of $2^m \cdot s$ natural numbers each of (binary) size at most m. The product of $2^m \cdot s$ natural numbers each of size at most m can be computed in $2^{\mathrm{O}(m \log s)}$ time and is of size $2^{\mathrm{O}(m \log s)}$. Now note that $\log p \geq r$ iff the integral part of the left-hand side is $\geq r$ (since the right-hand side is an integer), but the integral part of $\log p$ is just the length of the binary representation of p, which we have just computed.

To establish the EXPTIME upper bound, it remains to be shown that the numbers $a_i = |P^{-1}(\bar{o}_i)|$ can be computed in exponential time. This can be done by first computing the pushdown system corresponding to P, which is of size exponential in the size of P, and then invoking Theorem 1 to compute the set $\{(\bar{h}_0, F(\bar{h}_0)) \mid \bar{h}_0 \text{ is a high input}\}$. ◀

▶ Remark. The algorithm in the proof of Theorem 8 runs in time polynomial in the length of the program and exponential in the number of variables.

4 Information leakage from timing behavior

Let us now consider the question of estimating the information leaked by a program by its "timing behavior". We shall use the "number of steps" taken by a program as an abstraction of

its timing behavior. Given a program P, with high input variables $\bar{h}$ and low input variables $\bar{l}$, let $\text{Steps}_P : 2^{\bar{h}} \times 2^{\bar{l}} \to \mathbb{N}$ be the function such that $\text{Steps}_P(\bar{h}_0, \bar{l}_0)$ is the number of steps in the computation of $P(\bar{h}_0, \bar{l}_0)$. More precisely, this number is the number of steps in the corresponding computation of the pushdown system realizing the program P.

▶ **Definition 9.** A program P is *timing non-interferent* if the function Steps_P is non-interferent. Furthermore, if μ is the distribution on inputs to P, then $\text{SE}_\mu(\text{Steps}_P)$ is the *information leaked by the timing behavior* of P.

While programs. A terminating while program takes at most $\ell \cdot 2^n$ steps, where ℓ is the number of statements in the program and n is the total number of variables of the program (input, output and local). Hence, the running time of the program can be represented as a natural number whose (binary) size is polynomial in the size of program. Thus, we can easily modify the upper bound proof for deciding non-interference in while programs to the case of deciding timing non-interference in while programs. The lower bound proof for deciding non-interference in while programs can also be easily modified to give a lower bound on timing non-interference of while programs.

▶ **Lemma 10.** *Deciding timing non-interference for while programs with one high input, no low inputs and no outputs is* PSPACE*-hard. Deciding whether a while program is timing non-interferent can be done in* PSPACE.

Recursive programs. As in the case of while programs, the lower bound for deciding timing non-interference for recursive programs is a modification of the proof for Theorem 7.

▶ **Lemma 11.** *Deciding timing non-interference for recursive programs with one high input, no low inputs and no outputs is* EXPTIME*-hard.*

The upper bound proofs for bounding information leakage are more involved. The presence of recursion (i.e., the stack) implies that the length of the computation is no longer bounded by $\ell \cdot 2^n$ as in the case of while programs. Indeed, the length of a computation can be as high as doubly exponential, and the upper bound proof will depend on the ability to compute the length of a computation in exponential time. (Note that the length of a computation can be represented as an exponential-size number). In order to demonstrate this fact, we will establish some facts about deterministic pushdown systems.

Given a deterministic PDS $\mathcal{P}$, we say that a computation $c_0 \xrightarrow{\lambda_1} c_1 \cdots \xrightarrow{\lambda_m} c_m$ of $\mathcal{P}$ is *terminating* if there is no transition out of c_m. A state $q \in Q$ is a *good state* if there exists a terminating computation $c_0 \xrightarrow{\lambda_1} c_1 \cdots \xrightarrow{\lambda_m} c_m$ with $c_0 = (q, \varepsilon)$. We first establish that that the length of a computation from a good state is of at most exponential length.

▶ **Lemma 12.** *Let $\mathcal{P} = (Q, \Gamma, \delta)$ be a deterministic PDS and $q \in Q$ a good state. If there exists a configuration c with $(q, \varepsilon) \overset{m}{\Rightarrow}_{\mathcal{P}} c$, then $m \leq |Q| \cdot |\Gamma|^{|Q|+1}$.*

We now show that, even though the length of a computation of a deterministic pushdown system can be exponential, the length of the computation from a configuration (q, ε) to (q', ε) can be computed in polynomial time. This is proved by modifying the "summaries construction" algorithm used to decide reachability in pushdown systems [5]. We recall salient points of this algorithm before we prove the desired theorem.

The "summaries construction" algorithm proceeds iteratively, building an edge-labeled graph on the states of a pushdown system $\mathcal{P}$. At each step of the algorithm, edges are added and the algorithm terminates when a fixed point is reached. The set of labels on the edges is $\Gamma \cup \{\varepsilon\}$. Intuitively, the edge $q \overset{a}{\rightsquigarrow} q'$ means that there is a valid computation $(q, \varepsilon) \Rightarrow (q', a)$

of $\mathcal{P}$. The initial graph is constructed from the internal actions and the stack push transitions. New edges are constructed by taking the "transitive closure" of these edges with the stack pop transitions. For example, if $q \overset{a}{\rightsquigarrow} q'$ is an edge in the graph and $(q', a, q'') \in \delta_{\text{rtn}}$ then a new edge $q \overset{\varepsilon}{\rightsquigarrow} q''$ is added to the graph. We modify this algorithm by maintaining the execution time on the labels as well.

▶ **Theorem 13.** *There is a polynomial-time algorithm that, given a PDS $\mathcal{P} = (Q, \Gamma, \delta)$ and a set $Q_0 \subseteq Q$ of good states, outputs the set $\{(q, q', m) \mid q \in Q_0$ and $(q, \varepsilon) \overset{m}{\Rightarrow}_{\mathcal{P}} (q', \varepsilon)\}$.*

Proof. The algorithm constructs an edge-labeled directed graph $\mathcal{G}$ iteratively. The set of nodes of $\mathcal{G}$ is $\text{Reach}(Q_0) = \{q \mid \exists q_0 \in Q_0\, \exists w \in \Gamma^*\, (q_0, \varepsilon) \Rightarrow (q, w)\}$. Note that this set can be constructed in polynomial time thanks to Theorem 1. The set of labels on the edges of $\mathcal{G}$ is $\mathbb{N} \times (\Gamma \cup \{\varepsilon\})$. The graph $\mathcal{G}$ is constructed by computing a sequence of graphs $\mathcal{G}_0, \mathcal{G}_1, \ldots$ such that the set of edges of $\mathcal{G}_i$ is a subset of the set of edges of $\mathcal{G}_{i+1}$. The iteration terminates when $\mathcal{G}_i = \mathcal{G}_{i+1}$, in which case $\mathcal{G} = \mathcal{G}_i$. Initially, the set of edges in $\mathcal{G}_0$ is

$$\{(q \xrightarrow{(1,\varepsilon)} q_1) \mid (q, q_1) \in \delta_{\text{int}}\} \cup \{(q \xrightarrow{(1,a)} q_1 \mid (q, q_1, a) \in \delta_{\text{cll}}\}.$$

Assume now that $\mathcal{G}_i$ has been constructed. Then $\mathcal{G}_{i+1}$ is constructed as follows:

- for each pair of edges $q \xrightarrow{(m_1,\varepsilon)} q_1$ and $q_1 \xrightarrow{(m_2,a)} q_2$ in $\mathcal{G}_i$, we add the edge $q \xrightarrow{(m_1+m_2,a)} q_2$;
- for each pair of edges $q \xrightarrow{(m_1,a)} q_1$ and $q_1 \xrightarrow{(m_2,\varepsilon)} q_2$ and in $\mathcal{G}_i$, we add the edge $q \xrightarrow{(m_1+m_2,a)} q_2$;
- for each $a \in \Gamma$, each edge $q \xrightarrow{(m,a)} q_1$ in $\mathcal{G}_i$, and each transition $(q_1, a, q_2) \in \delta_{\text{rtn}}$, we add the edge $q \xrightarrow{(m+1,\varepsilon)} q_2$.

Once $\mathcal{G}$ has been constructed, the algorithm outputs the set

$$\{(q, q, 0) \mid q \in Q_0\} \cup \{(q, q', m) \mid q \in Q_0 \text{ and } q \xrightarrow{(m,\varepsilon)} q' \text{ is an edge of } \mathcal{G}\}.$$

We claim that:

1. The algorithm terminates in polynomial time.
2. The output equals $\{(q, q', m) \mid q \in Q_0$ and $(q, \varepsilon) \overset{m}{\Rightarrow} (q', \varepsilon)\}$. ◀

▶ **Theorem 14.** *The problem of deciding whether the information leaked by the timing behavior of a recursive program P does not exceed q is in* EXPTIME.

Proof. As in the case of the proof of Theorem 8, we can assume that P has no low inputs. We can construct the pushdown system corresponding to P and, using Theorem 13, compute the set $R = \{(\bar{h}_0, \text{Steps}_P(\bar{h}_0)) \mid \bar{h}_0$ is a high input$\}$. Now we can partition the set of inputs according to the equivalence relation $\equiv$ defined by $\bar{h}_1 \equiv \bar{h}_2$ iff $\text{Steps}_P(\bar{h}_1) = \text{Steps}_P(\bar{h}_2)$. Let $a_1, \ldots, a_k$ be the partition sizes of $\equiv$. Note that these partition sizes can be computed in time polynomial in the size of the set R, i.e. exponential in the size of P. If m is the number of input variables, then $\sum_1^k a_i = 2^m$ and $\text{SE}_{\mathsf{U}}(\text{Steps}_P) \leq q$ iff $\sigma(a_1, \ldots, a_k) \geq 2^m(m - q)$. (Recall that $\sigma(a_1, \ldots, a_k) = \sum_{i=1}^k a_i \log a_i$.) The latter can now be decided in exponential time as in the proof of Theorem 8. ◀

5 Conclusions and future work

We have considered the problems of checking non-interference and of bounding information leakage in (deterministic) recursive boolean programs with uniformly distributed inputs, proving both problems to be EXPTIME-complete. This implies an EXPTIME upper bound for non-recursive programs, which improves the previously known upper bounds. For the special

case when the number of outputs and low inputs is logarithmic in the size of the program, we have established a tight PSPACE upper bound for non-recursive programs.

We have also considered the problem of checking non-interference and of bounding information leakage in recursive boolean programs when the attacker observes the number of execution steps of the program (and not the explicit outputs). Once again, our problems turn out to be EXPTIME-complete in this setting. The proof of the upper bound is interesting from a practical standpoint as we have shown that existing algorithms used for analyzing safety properties in recursive programs can be used for computing information leakage. In fact, we are currently working on a BDD-based symbolic algorithm for computing information leakage in recursive programs.

We have used measures based on Shannon's entropy and mutual information. Nevertheless, our techniques are useful for computing information leakage with respect to other measures. For example, if we use min-entropy to define mutual information [31, 16], the problem of bounding information leakage (from explicit outputs or from timing behavior) for programs with uniformly distributed high inputs is again EXPTIME-complete for recursive programs. We believe that the techniques used in this paper will also be useful for other scenarios, such as the case when we are interested in only the amount of information leaked about certain selected bits of the input.

In addition to extending the results to other scenarios as described above, one particular open problem is to close the gap between the lower bound (PSPACE) and the upper bound (EXPTIME) for non-recursive programs with no restrictions on the number of inputs and outputs. Another interesting direction for future research is to extend our results to programs with probabilistic choices.

Acknowledgements. This research was done while Rohit Chadha was at LSV, ENS Cachan and INRIA Saclay. Michael Ummels was supported by the DFG projects PROBFOR, SYANCO and QuaOS.

References

1 J. Agat. Transforming out timing leaks. In *POPL '00*, pages 40–53, 2000.
2 E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009.
3 M. Backes, B. Köpf, and A. Rybalchenko. Automatic discovery and quantification of information leaks. In *IEEE Symposium on Security and Privacy*, pages 141–153, 2009.
4 G. Barthe, T. Rezk, and M. Warnier. Preventing timing leaks through transactional branching instructions. *Electronic Notes in Theoretical Computer Science (Proc. QAPL '06)*, 153(2):33–55, 2006.
5 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97*, pages 135–150, 1997.
6 D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
7 P. Černý, K. Chatterjee, and T. A. Henzinger. The complexity of quantitative information flow problems. In *CSF '11*, pages 205–217, 2011.
8 R. Chadha and M. Ummels. The complexity of quantitative information flow in recursive programs. Technical Report LSV-12-15, ENS Cachan, 2012.
9 K. Chatzikokolakis, T. Chothia, and A. Guha. Statistical measurement of information leakage. In *TACAS '10*, pages 390–404, 2010.
10 K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Probability of error in information-hiding protocols. In *CSF '07*, pages 341–354, 2007.

11 K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Anonymity protocols as noisy channels. *Information and Computation*, 206(2-4), 2008.

12 D. Clark, S. Hunt, and P. Malacaria. Quantified interference for a while language. *Electronic Notes in Theoretical Computer Science (Proc. QAPL '04)*, 112:49–166, 1984.

13 D. Clark, S. Hunt, and P. Malacaria. Quantitative information flow, relations and polymorphic types. *Journal of Logic Computation*, 15(2):181–199, 2005.

14 D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

15 D. E. R. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

16 Y. Dodis, R.Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal of Computing*, 38(1):97–139, 2008.

17 C. Flanagan and J. B. Saxe. Avoiding exponential explosion: Generating compact verification conditions. In *POPL '01*, pages 193–205, 2001.

18 J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

19 J. W. Gray III. Toward a mathematical foundation for information flow security. In *IEEE Symposium on Security and Privacy*, pages 21–35, 1991.

20 D. Hedin and D. Sands. Timing aware information flow security for a JavaCard-like bytecode. *Electronic Notes in Theoretical Computer Science*, 141(1):163–182, 2005.

21 A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. In *Proceedings of the USSR Academy of Sciences*, 145, pages 293–294, 1962.

22 B. Köpf and D. A. Basin. An information-theoretic model for adaptive side-channel attacks. In *ACM Conference on Computer and Communications Security*, pages 286–296, 2007.

23 B. Köpf and M. Dürmuth. A provably secure and efficient countermeasure against timing attacks. In *CSF '09*, pages 324–335, 2009.

24 B. Köpf and A. Rybalchenko. Approximation and randomization for quantitative information-flow analysis. In *CSF '10*, pages 3–14, 2010.

25 J. K. Millen. Covert channel capacity. In *IEEE Symposium on Security and Privacy*, pages 60–66, 1987.

26 S. S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 1997.

27 A. D. Pierro, C. Hankin, and H. Wiklicky. Quantifying timing leaks and cost optimisation. In *ICICS '08*, pages 81–96, 2008.

28 T. W. Reps, S. Horwitz, and S. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL '95*, pages 49–61, 1995.

29 J. C. Reynolds. Syntactic control of interference. In *POPL '78*, pages 39–46, 1978.

30 C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

31 G. Smith. On the foundations of quantitative information flow. In *FOSSACS '09*, pages 288–302, 2009.

32 R. van der Meyden and C. Zhang. Algorithmic verification of noninterference properties. *Electronic Notes in Theoretical Computer Science*, 168:61–75, 2007.

33 H. Yasuoka and T. Terauchi. On bounding problems of quantitative information flow. In *ESORICS '10*, pages 357–372, 2010.

34 H. Yasuoka and T. Terauchi. Quantitative information flow - verification hardness and possibilities. In *CSF '10*, pages 15–27, 2010.

35 H. Yasuoka and T. Terauchi. Quantitative information flow as safety and liveness hyperproperties. In *QAPL 2012*, pages 77–91, 2012.

Computationally Complete Symbolic Attacker in Action

Gergei Bana[*1], Pedro Adão[†2], and Hideki Sakurada[3]

1 **MSR-INRIA Joint Centre, Palaiseau, France** bana@math.upenn.edu
2 **SQIG-IT and IST-TULisbon, Portugal,** pedro.adao@ist.utl.pt
3 **NTT Communication Science Laboratories, Atsugi, Kanagawa, Japan,** sakurada.hideki@lab.ntt.co.jp

Abstract

We show that the recent technique of computationally complete symbolic attackers proposed by Bana and Comon-Lundh [7] for computationally sound verification of security protocols is powerful enough to verify actual protocols. In their work, Bana and Comon-Lundh presented only the general framework, but they did not introduce sufficiently many axioms to actually prove protocols.

We present a set of axioms—some generic axioms that are computationally sound for all PPT algorithms, and two specific axioms that are sound for CCA2 secure encryptions—and illustrate the power of this technique by giving the first computationally sound verification (secrecy and authentication) via symbolic attackers of the NSL Protocol that does not need any further restrictive assumptions about the computational implementation. The axioms are entirely modular, not particular to the NSL protocol.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Security Protocols, Symbolic Adversary, Computational Soundness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.546

1 Introduction

Research into computational soundness of symbolic verification of security protocols started with Abadi and Rogaway [1], followed by many others for passive [12, 10, 2] and active adversaries. The aim is always that symbolic proofs imply computational security. Works concerning active adversaries can be divided into two groups. Works in one [3, 16, 4, 14] define symbolic adversaries, and soundness theorems state that under certain circumstances, if there is no successful symbolic attack, then there is no successful computational attack either. The other group aims to work directly in the computational model [18, 8, 13, 11]. This paper focuses on the first.

The first group, where symbolic attacker is defined, gives hope that already existing automated tools may be used for computationally sound verification, but these soundness theorems require large sets of restrictive assumptions on computational implementations. Typically they assume that no key cycles can ever be created, that bit strings can be unambiguously parsed into terms, that there is no dynamic corruption, that keys are certified

* Partially supported by the ANR project ProSe and FCT project ComFormCrypt PTDC/EIA-CCO/113033/2009.
† Partially supported by FCT projects ComFormCrypt PTDC/EIA-CCO/113033/2009 and PEst-OE/EEI/LA0008/2011.

(badly generated keys cannot be used), etc. These assumptions as well as reasons why they are not realistic are discussed in [15].

Recently, Bana and Comon-Lundh presented in [7] (and improved in [6]) a new technique to define symbolic attackers. They called this new symbolic adversary *computationally complete symbolic adversary*, as it is capable of doing everything that a computational adversary is capable of without any restrictions. Instead of listing every kind of move a symbolic adversary is allowed to do as usual with Dolev-Yao adversaries, a few rules (axioms) are listed that the symbolic adversary is not allowed to violate. That is, the symbolic adversary is allowed to do everything that is consistent with these axioms. The axioms are first-order formulas expressing properties that hold computationally for PPT adversaries and for the computational implementations of cryptographic primitives. Their main result is that once it is shown that no successful symbolic adversary can exist complying some set of axioms, then for any computational implementation satisfying that set of axioms (that is, implementations for which the axioms are computationally sound), successful computational attacks are impossible as long as the number of sessions is bounded in the security parameter.

In their original work however, they did not show that their technique could actually be used for practical protocol verification as they only presented the general framework and a few computationally sound axioms as a proof of concept. To actually prove protocols, more axioms have to be introduced in order to *weaken* the symbolic adversary sufficiently close to the computational adversary (with unrealistically strong symbolic attackers, no protocol can be verified).

In this paper, we show the technique of Bana and Comon-Lundh can indeed be used for protocol verification. We introduce some modular, computationally sound axioms, and then illustrate that the technique with the axioms we introduced can be used to verify secrecy and authentication of an actual protocol, namely, the Needham-Schroeder-Lowe protocol. More precisely, we show that there is no symbolic adversary for which the violation of either secrecy or authentication (or both) of the NSL protocol is consistent with the set of general axioms we give together with an additional minimal parsing property that the computational implementation of *pairing* must satisfy (otherwise there is an attack). Applying the main theorem of [7], this means that there is no computational adversary of the NSL protocol (in an implementation satisfying the axioms) that can violate secrecy or authentication with non-negligible probability.

The set of axioms we give is divided into four groups. One has a number of general axioms sound for any computational implementation. Then, there is a group consisting of the equations required for function symbols, such as the decryption of a cipher gives the plaintext back. Then, there is a group of two axioms sound for CCA2 encryptions, one expressing the secrecy of a CCA2 encrypted item, and one expressing the non-malleability property of CCA2 encryptions. Furthermore, to prove security of the NSL protocol, one more property is needed expressing a certain parsing unambiguity, which needs to be assumed as otherwise an attack exists.

The axioms are not particular to NSL protocols. They are modular. Introducing further primitives will not destroy the soundness of these axioms, they do not have to be proven again.

The technique of [7] allows to avoid *all* restrictions mentioned before on the computational world. *Once a protocol is proven secure in our symbolic model with respect to a set of axioms, then all properties that the computational implementation has to satisfy for computational security are included in the axioms.* Any number of bad keys are allowed to be generated by the adversary; any number of corrupted, uncorrupted, or dynamically

corrupted parties can be present. As for parsing of bit strings into terms, previous soundness results relied on unambiguous parsing. Within this framework, we do not need such an assumption. If unambiguous parsing is needed for the security of a protocol, then it is necessary to list it as a property that secure implementations need to satisfy. The only needed assumption for proving NSL is that an honestly generated nonce N cannot be non-negligibly parsed into a pair, such that the second part of the pair is some (dishonest) agent name, *i.e.*, looks like $\langle n, Q\rangle$ for some n. This is a necessary assumption, as the failure of it results in an attack, presented in [9]. This can easily be achieved in an implementation by, for example, checking the length of bit strings that should correspond to nonces. Other than this, no parsing hypothesis is assumed. For example, honestly generated nonces may collide with other kinds of pairs, encryptions could *a priori* collide with other kinds of expressions, etc. That is, *tagging of pairs, encryption etc is not necessary for the security of the NSL protocol.*

In fact, the security proof is long exactly because of parsing ambiguities as any term that was created by an honest agent or the adversary may *a priori* be wrongly parsed by another honest agent. The fact that they are not wrongly parsed had to be derived in the protocol proof from our axioms, that is, from reasons other than tagging. Had we assumed tagging and completely unambiguous parsing (which, in fact, has always been assumed by earlier NSL proofs, even in Cryptoverif), the proof would have been quite short.

In [20] the author provides a direct computational proof that the NSL protocol is secure if the encryption scheme used is IND-CCA. His proof also uses bounded number of sessions, but implicitly assumes some level of unambiguous parsing (hence did not find the attack presented here). However, it allows agents to run both roles (but not against themselves).

We would like to emphasize that our aim here is to demonstrate that the technique works, and not to provide the most general possible verification for the NSL protocol. Further generalizations are possible at the cost of much longer proofs. For example, in the current proof, we assume that each honest agent only executes either the initiator or the responder role as this makes the proof much shorter and clearer. We have however been able to complete the proof for the case when they are allowed to run both sessions, even against themselves at the cost of an additional parsing axiom. A further assumption is that triples are created from pairs. It is possible to do the proofs without this assumption, and have a separate function symbol for triples (and introduce further necessary requirements to avoid attacks), but again, it would make the proofs far longer.

The contributions of this paper include (i) the set of general axioms that are computationally sound for any PPT implementation, (ii) the non-malleability axiom that is computationally sound for CCA2 security, (iii) the additional parsing axiom needed to avoid an attack to the NSL, and (iv) the security proof itself. Again, the axioms are all modular, independent of the protocol, and they do not have to be proved again if further primitives are included.

This paper is organized as follows: we start by recalling the framework of [7] (Section 2). In Section 3, we show how the NSL protocol and its execution can be formulated in the proposed framework. In Section 4, we present the first contribution of this paper introducing the set of computational sound axioms that are sufficient to show both secrecy and authentication of the NSL protocol. In Section 5, we show our new computational attack to the NSL. In Section 6, we show a few simple examples of how inconsistency of formulas can be proven in the framework. In Section 7, we prove that no symbolic adversary compliant with the presented axioms can violate secrecy or authentication of the NSL protocol. In Section 8, we summarize our results and present directions for future work.

2 Symbolic Execution and Properties

2.1 Terms, Frames, and Formulas

Terms are built from a set of function symbols $\mathcal{F}$ containing a countable set of *names* $\mathcal{N}$, a countable set of *handles* $\mathcal{H}$, which are of 0-arity, and a finite number of higher arity symbols. Names denote items honestly generated by agents, while handles denote inputs of the adversary. Let $\mathcal{X}$ be an infinite set of *variables*. A *ground term* is a term without variables. *Frames* are sequences of terms together with name binders: a frame ϕ can be written $(\nu\overline{n}).p_1 \mapsto t_1, \ldots, p_n \mapsto t_n$ where $p_1, \ldots p_n$ are place holders that do not occur in $t_1, \ldots, t_n$ and $\overline{n}$ is a sequence of names. $\mathsf{fn}(\phi)$, the *free names* of ϕ are names occurring in some t_i and not in $\overline{n}$. The *variables* of ϕ are the variables of $t_1, \ldots, t_n$.

Let $\mathcal{P}$ be a set of predicate symbols over terms. $\mathcal{P}$ contains the binary predicate $=$ used as $t_1 = t_2$, and a family of $n+1$-arity predicates $\vdash_n$ used as $t_1, ..., t_n \vdash t$, intended to model the adversary's capability to derive something. We drop the index n for readability. We *allow any FOL interpretation* of the predicates that does not contradict our axioms, which we will introduce later.

Let $\mathcal{M}$ be *any first-order structure* that interprets the function and predicate symbols. We only require that $\mathcal{M}$ interprets terms and predicates such that $=$ is interpreted as the equality in the underlying domain $D_{\mathcal{M}}$. Given an assignment σ of elements in $D_{\mathcal{M}}$ to the free variables of term t, we write $[\![t]\!]^{\sigma}_{\mathcal{M}}$ for the interpretation of $t\sigma$ in $\mathcal{M}$ ($[\![\_]\!]^{\sigma}_{\mathcal{M}}$ is the unique extension of σ into a homomorphism of $\mathcal{F}$-algebras). For any first order structure $\mathcal{M}$ over $\mathcal{F}$ and $\mathcal{P}$, and any assignment σ of the free variables of θ in the domain of $\mathcal{M}$, the satisfaction relation $\mathcal{M}, \sigma \models \theta$, is defined as usual in FOL.

2.2 Symbolic Execution of a Protocol

▶ **Definition 2.1.** A *symbolic state* of the network consists of:

- a control state $q \in Q$ together with a sequence of names $n_1, \ldots, n_k$ (so far generated)
- a sequence constants called *handles* $h_1, \ldots, h_n$ (recording the attacker's inputs)
- a ground frame ϕ (the agents' outputs)
- a set of formulas Θ (the conditions that have to be satisfied in order to reach the state).

A *symbolic transition sequence* of a protocol Π is a sequence

$$(q_0(\overline{n_0}), \emptyset, \phi_0, \emptyset) \to \ldots \to (q_m(\overline{n_m}), \langle h_1, \ldots, h_m \rangle, \phi_m, \Theta_m)$$

where, for every $m - 1 \geq i \geq 0$, there is a transition rule $(q_i(\overline{\alpha_i}), q_{i+1}(\overline{\alpha_{i+1}}), \langle x_1, \ldots, x_i \rangle, x, \psi, s)$ such that $\overline{n} = \overline{\alpha_{i+1}} \setminus \overline{\alpha_i}$, $\phi_{i+1} = (\nu\overline{n}).(\phi_i \cdot p \mapsto s\rho_i\sigma_{i+1})$, $\overline{n_{i+1}} = \overline{n_i} \uplus \overline{n}$, $\Theta_{i+1} = \Theta_i \cup \{\phi_i \vdash h_{i+1}, \psi\rho_i\sigma_{i+1}\}$ where $\sigma_{i+1} = \{x_1 \mapsto h_1, \ldots, x_{i+1} \mapsto h_{i+1}\}$ and ρ_i is a renaming of the sequence $\overline{\alpha_i}$ into the sequence $\overline{n_i}$. We assume a renaming that ensures the freshness of the names $\overline{n}$: $\overline{n} \cap \overline{n_i} = \emptyset$.

▶ **Definition 2.2.** Given an interpretation $\mathcal{M}$, a transition sequence of protocol Π

$$(q_0(\overline{n_0}), \emptyset, \phi_0, \emptyset) \to \ldots \to (q_m(\overline{n_m}), \langle h_1, \ldots, h_m \rangle, \phi_m, \Theta_m)$$

is *valid w.r.t.* $\mathcal{M}$ if, for every $m - 1 \geq i \geq 0$, $\mathcal{M} \models \Theta_{i+1}$.

Initialization. For technical purposes, we assume that all honestly generated items (nonces, random inputs of encryptions, etc) are generated upfront. This is possible as we assumed bounded number of sessions. We always set $\phi_0 = \nu\overline{n}()$, where $\overline{n}$ are the honestly generated items. ϕ_1 contains the output of the initialization, that is, the names and the public keys.

2.3 Predicates, Constraints and FOL Formulas in Executions

$\mathcal{M}$ modeled, among others, the predicate $t_1, ..., t_n \vdash t$. In *executions* however, instead of this predicate, we consider a predicate that we write as $\hat{\phi}, t_1, ..., t_n \vdash t$. This is also an $n+1$-arity predicate. $\hat{\phi}$ is just a symbol, not an argument, and it represents the frame containing the messages that protocol agents sent out, that is, the information available from the protocol to the adversary. Computational semantics of the predicates $x = y$ and $\hat{\phi}, x_1, ..., x_m \vdash x$ was defined in [7] and improved in [6]. Here we just briefly mention that $=$ refers to equality up to negligible probability, and $\vdash$ means that the adversary is able to compute (with a PT algorithm) the right side from the left. We also use another predicate, $W(x)$, which just tells if x is the name of an agent. We also use various *constraints*: $\mathsf{Handle}(h)$ means h is a handle; $\mathsf{RandGen}(x)$ means that x was honestly, randomly generated (i.e. appearing under ν in the frame); $x \sqsubseteq \hat{\phi}$ means that x was part of a message sent out by an agent (i.e. listed in the frame ϕ); $x \sqsubseteq \vec{x}$ means x is part of $\vec{x}$. $dK \sqsubseteq_{\not d} \hat{\phi}$ means dK occurs other than in a decryption position $dec(\ , dK)$ in ϕ, and $dK \sqsubseteq_{\not d} \vec{x}$ is analogous. Let us introduce the following abbreviations:

- $x \sqsubseteq \hat{\phi}, \vec{x} \quad \equiv \quad x \sqsubseteq \hat{\phi} \vee x \sqsubseteq \vec{x}$
- $x \sqsubseteq_{\not d} \hat{\phi}, \vec{x} \quad \equiv \quad x \sqsubseteq_{\not d} \hat{\phi} \vee x \sqsubseteq_{\not d} \vec{x}$
- $\mathsf{fresh}(x; \hat{\phi}, \vec{x}) \quad \equiv \quad \mathsf{RandGen}(x) \wedge x \not\sqsubseteq \hat{\phi}, \vec{x}$
- $\vec{x} \preccurlyeq \hat{\phi} \quad \equiv \quad h \sqsubseteq \vec{x} \wedge \mathsf{Handle}(h) \rightarrow \hat{\phi} \vdash h$

Given a first-order model $\mathcal{M}$ as before, satisfaction of predicates and constraints in a *symbolic* execution is defined as:

- Interpretation of predicates by $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, (n_1, \ldots, n_k)$, where σ is a substitution as above, $t_1, ..., t_m$ are closed terms, and $n_1, ..., n_k$ are names: (note the interpretation depends on $\mathcal{M}$) is defined as follows
  - $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, (n_1, \ldots, n_k) \models t = t'$ if $\mathcal{M}, \sigma \models t = t'$
  - $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, \overline{n} \models \hat{\phi}, s_1, \ldots, s_n \vdash t$ if $\mathcal{M}, \sigma \models s_1, \ldots, s_n, t_1, \ldots, t_m \vdash t$.
  - $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, \overline{n} \models W(x)$ if $\mathcal{M}, \sigma \models W(x)$
- Interpretations of constraints by $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, (n_1, \ldots, n_k)$, where σ is a substitution as above, $t_1, ..., t_m$ are closed terms, and $n_1, ..., n_k$ are names: (do not depend on the model $\mathcal{M}$):
  - $\mathsf{Handle}(h)$ for h closed term:
    $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, (n_1, \ldots, n_k) \models \mathsf{Handle}(h)$ if $h \in \mathcal{H}$.
  - $\mathsf{RandGen}(s)$ for s closed term:
    $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, (n_1, \ldots, n_k) \models \mathsf{RandGen}(s)$ if $s \in \mathcal{N}$ and $\mathcal{M}, \sigma \models s = n_1 \vee \ldots \vee s = n_k$.
  - $t \sqsubseteq \hat{\phi}$, where t is closed term:
    $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, \overline{n} \models t \sqsubseteq \hat{\phi}$ if t is a subterm of some t_i
  - $t \sqsubseteq s_1, ..., s_n$, where $s_1, ..., s_n$ and t are closed terms:
    $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, \overline{n} \models t \sqsubseteq s_1, ..., s_n$ if t is a subterm of some s_i
- Interpretation of any FOL formula in which there are no free variables under constraints by $\mathcal{M}, \sigma, \langle t_1, \ldots, t_m \rangle, (n_1, \ldots, n_k)$ where σ is a substitution as above, is defined recursively as:
  - Interpretations of $\theta_1 \wedge \theta_2$, $\theta_1 \vee \theta_2$, and $\neg\theta$ are defined as usual in FOL
  - If x is not under a constraint in θ, interpretations of $\forall x\theta$ and $\exists x\theta$ are defined as usual in FOL.
  - If x occurs under a constraint in θ, then
    - $\mathcal{M}, \sigma, \langle t_1, \ldots, t_n \rangle, (n_1, \ldots, n_k) \models \forall x\theta$ iff for every ground term t,
      $\mathcal{M}, \sigma, \langle t_1, \ldots, t_n \rangle, (n_1, \ldots, n_k) \models \theta\{x \mapsto t\}$

  - $\mathcal{M}, \sigma, \langle t_1, \ldots, t_n \rangle, (n_1, \ldots, n_k) \models \exists x \theta$ iff there is a ground term t, $\mathcal{M}, \sigma, \langle t_1, \ldots, t_n \rangle, (n_1, \ldots, n_k) \models \theta\{x \mapsto t\}$
- Satisfaction at step m: $\mathcal{M}, (q, \langle h_1, \ldots, h_m \rangle, \overline{n}, \phi_m, \Theta) \models \theta$ iff $\mathcal{M}, \phi_m, \overline{n} \models \theta$.

3 The NSL Protocol and Its Symbolic Execution

We now formulate the NSL protocol and its execution in the above framework. The steps of the protocol, as usual are

1. $A \to B \ : \ \{N_1, A\}_{eK_B}$ 2. $B \to A \ : \ \{N_1, N_2, B\}_{eK_A}$ 3. $A \to B \ : \ \{N_2\}_{eK_B}$

We use a randomized public-key encryption symbol: $\{m\}^r_{eK_Q}$ is intended to represent the encryption of the plaintext m with the public-key of the principal Q, with a random seed r. So, consider the set of constructors $\mathcal{F}_c = \{\{\_\}_\_, \langle \_, \_ \rangle, e\_, d\_, K\_\}$, and the set of destructors $\mathcal{F}_d = \{dec(\_, \_), \pi_1(\_), \pi_2(\_)\}$, with the following equations:

- Decryption of an encryption returns the plaintext: $dec(\{x\}^R_{eK}, dK) = x$
- First projection of pairing: $\pi_1(\langle x, y \rangle) = x$, second projection of pairing: $\pi_2(\langle x, y \rangle) = y$

We will use pairs to construct triples: $\langle x, y, z \rangle \equiv \langle x, \langle y, z \rangle \rangle$. From now on, constant symbols as h_i and R_i will be used as meta-symbols, they are not the actual elements of $\mathcal{N}$ or $\mathcal{H}$.

We define the roles of principals as follows: the initiator, communicating with intended party Q, does the following sequence of steps in session i (denoted by $Init^A_{NSL}[A, i, Q, N_1, h_1, h_3, R_1, R_3]$):

1. Receives handle h_1 that triggers the start of the session with intended party Q;
2. A generates nonce N_1;
3. A sends $\{N_1, A\}^{R_1}_{eK_Q}$;
4. A receives h_3, and checks: **a.** $\pi_1(dec(h_3, dK_A)) = N_1$ **b.** $\pi_2(\pi_2(dec(h_3, dK_A))) = Q$;
5. A sends $\{\pi_1(\pi_2(dec(h_3, dK_A)))\}^{R_3}_{eK_Q}$;
6. A sends $c_i\big(A, Q, N_1, \pi_1(\pi_2(dec(h_3, dK_A)))\big)$.

For verification purposes, let c_i be a special function symbol, that takes as arguments A, B, N_1, N_2, respectively who commits for whom and the corresponding nonces. $c_i(A, B, N_1, N_2)$ is sent along with $\{N_1, N_2, B\}_A$. For the responder, there is a similar commitment: at the end of the protocol, B emits (as a last message) $c_r(A, B, N_1, N_2)$.

The responder does the following sequence of steps in session i' which we denote informally by $Resp^B_{NSL}[B, i', N_2, h_2, h_4, R_2]$:

1. B receives some h_2 from the adversary and checks:
   - $W(\pi_2(dec(h_2, dK_B)))$ (Checks that it is a name of someone);
2. B generates nonce N_2;
3. B sends $\{\pi_1(dec(h_2, dK_B)), N_2, B\}^{R_2}_{eK_{\pi_2(dec(h_2, dK_B))}}$;
4. B receives h_4, and checks if $dec(h_4, dK_B) = N_2$;
5. B sends $c_r(\pi_2(dec(h_2, dK_B)), B, \pi_1(dec(h_2, dK_B)), N_2)$.

▶ **Example 3.1.** We show the beginning of a possible branch in the symbolic execution of a single session of the NSL protocol.

$$(q_0, \emptyset, \phi_0, \emptyset) \bullet \longrightarrow \bullet\ (q_1, H_1, \phi_1, \Theta_1) \bullet \longrightarrow \bullet\ (q_2, H_2, \phi_2, \Theta_2) \bullet \longrightarrow \bullet\ (q_3, H_3, \phi_3, \Theta_3) \bullet \longrightarrow \bullet\ (q_4, H_4, \phi_4, \Theta_4)$$

where $\overline{n} = N_1, N_2, R_1, R_2, R_3$ and, with q^A_j, q^B_j counting the states of the A and B, $q_0 = (q^A_0, q^B_0)(\overline{n})$, $q_1 = (q^A_1, q^B_0)(\overline{n})$, $q_2 = (q^A_1, q^B_1)(\overline{n})$, $q_3 = (q^A_2, q^B_1)(\overline{n})$ and $q_4 = (q^A_2, q^B_2)(\overline{n})$. In other words, we interleave the actions of A and B, as in an expected execution and assume that the two processes were first activated.

- $\phi_0 = \nu_{K_A K_B AB}()$, $\Theta_0 = \emptyset$
- $H_1 = \emptyset$, $\phi_1 = \nu_{K_A K_B AB}(p_1 \mapsto (A, B, eK_A, eK_B))$, $\Theta_1 = \emptyset$
- $H_2 = \langle h_1 \rangle$, ϕ_2 extends ϕ_1 with $p_1 \mapsto \{\langle N_1, A\rangle\}^{R_1}_{eK_B}$, $\Theta_2 = \{\phi_1 \vdash h_1\}$
- $H_3 = \langle h_1, h_2 \rangle$, ϕ_3 extends ϕ_2 with $p_2 \mapsto \{\langle \pi_1 (dec(h_2, dK_B)), \langle N_2, B\rangle\rangle\}^{R_2}_{eK_{\pi_2(dec(h_2, dK_B))}}$, $\Theta_3 = \Theta_2 \cup \{\phi_2 \vdash h_2, W(\pi_2 (dec(h_2, dK_B)))\}$
- $H_4 = \langle h_1, h_2, h_3 \rangle$, ϕ_4 extends ϕ_3 with $p_3 \mapsto \{\pi_1 (\pi_2 (dec(h_3, dK_A)))\}^{R_3}_{eK_B}$, $\Theta_4 = \Theta_3 \cup \{\phi_3 \vdash h_3, \pi_1 (dec(h_3, dK_h)) = N_1, \pi_2 (\pi_2 (dec(h_3, dK_A))) = B\}$,
- $H_5 = \langle h_1, h_2, h_3, h_4 \rangle$, $\phi_5 = \phi_4$, $\Theta_5 = \Theta_4 \cup \{\phi_4 \vdash h_4, dec(h_4, dK_B) = N_2\}$,

Let $\mathcal{M}$ be a model such that $\pi_2 (dec(h_2, dK_B)) = A$,

$$h_2 =_{\mathcal{M}} \{\langle N_1, A\rangle\}^{R_1}_{eK_B}, \qquad h_3 =_{\mathcal{M}} \{\langle N_1, \langle N_2, B\rangle\rangle\}^{R_2}_{eK_A}, \qquad h_4 =_{\mathcal{M}} \{N\}^{R_3}_{eK_B},$$

and $\vdash_{\mathcal{M}}$ is simply the classical Dolev-Yao deduction relation. Then the execution sequence above is valid w.r.t. $\mathcal{M}$, and this corresponds to the correct execution of the NSL protocol between A and B.

▶ **Example 3.2.** Consider again Example 3.1, and a model $\mathcal{M}$ in which $N_0, \{N_1, N_2, B\}^{R_2}_{eK_A} \vdash_{\mathcal{M}} \{N_1, N_0, B\}^{r}_{eK_A}$ for an honestly generated nonce N_0 that can be chosen by the attacker; the transition sequence of the previous example is also valid w.r.t. this model. This however yields an attack, using a malleability property of the encryption scheme. Discarding such attacks requires some properties of the encryption scheme (for instance IND-CCA). It can be ruled out by the non-malleability axiom that we will introduce. From this example, we see that unexpected attacks can be found when some assumption is not explicitly stated as an axiom to limit adversarial capabilities.

4 The Axioms

This section contains the core results of this paper: a set of computationally sound axioms that are sufficient to prove security of actual protocols that use CCA2 secure encryptions. The axioms are not at all special to the NSL protocol and can be used in other protocol proofs too. They are entirely modular, so introducing further primitives will not invalidate their soundness, they do not have to be verified again. As usual, unquantified variables are universally quantified.

- **Equality is a Congruence.** The first axiom says that the equality is a congruence relation:
  - $x = x$, and the substitutability (congruence) property of equal terms holds for predicates (but not necessarily constraints).

  This axiom is computationally sound simply as we limit ourself to consider computational interpretations of predicates that are invariant if we change the arguments on sets with negligible probability. The computational interpretations of $=$, $\vdash$ and W are all such.
- **Axioms for the Derivability Predicate.** The following axioms are trivially computationally sound for what the PPT adversary can compute. The last is sound as we assume that all function symbols are interpreted as PT computable functions.
  - Self derivability: $\hat{\phi}, \vec{x}, x \vdash x$
  - Increasing capabilities: $\hat{\phi}, \vec{x} \vdash y \longrightarrow \hat{\phi}, \vec{x}, x \vdash y$
  - Commutativity: If $\vec{x}'$ is a permutation of $\vec{x}$, then $\hat{\phi}, \vec{x} \vdash y \longrightarrow \hat{\phi}, \vec{x}' \vdash y$
  - Transitivity of derivability: $\hat{\phi}, \vec{x} \vdash \vec{y} \wedge \hat{\phi}, \vec{x}, \vec{y} \vdash \vec{z} \longrightarrow \hat{\phi}, \vec{x} \vdash \vec{z}$
  - Functions are derivable: $\hat{\phi}, \vec{x} \vdash f(\vec{x})$

- **Axioms for Randomly Generated Items.** These are relations involving $\mathsf{RandGen}()$ and the $\sqsubseteq$ constraints that are not purely symbolic (for example, as $\sqsubseteq$ is a constraint, $x \sqsubseteq \hat{\phi}, x$ holds purely symbolically, so it does not have to be listed as an axiom). No telepathy expresses that randomly generated items not yet sent out are not guessable. It is sound as we assumed that random generation happens in a large enough space such that guessing is only possible with negligible probability. The second axiom is sound because random generation is independent of everything else, so a randomly generated item x given to the adversary cannot help to compute y from which it is independent, before x was sent out (that is, appear in ϕ). Once x appears in the frame (as e.g. $\{y\}^R_x$), giving x to the adversary may help to compute y. The condition $\vec{x}, y \preccurlyeq \hat{\phi}$ ensures that $\vec{x}, y$ do not contain future knowledge of the adversary in the form of handles computed later.
  - No telepathy: $\mathsf{fresh}(x; \hat{\phi}) \longrightarrow \hat{\phi} \nvdash x$
  - Fresh items are independent and hence contain no information about other items:

$$\mathsf{fresh}(x; \hat{\phi}, \vec{x}, y) \ \wedge \ \vec{x} \preccurlyeq \hat{\phi} \ \wedge \ y \preccurlyeq \hat{\phi} \ \wedge \ \hat{\phi}, \vec{x}, x \vdash y \ \longrightarrow \ \hat{\phi}, \vec{x} \vdash y$$

- **Equations for the fixed function symbols** discussed earlier:
  - $dec(\{x\}^R_{eK}, dK) = x; \quad \pi_1(\langle x, y\rangle) = x; \quad \pi_2(\langle x, y\rangle) = y$
- **Special to IND-CCA2 Encryption** Let $x_1, ..., x_n \preccurlyeq \hat{\phi} \ \equiv \ x_1 \preccurlyeq \hat{\phi} \wedge ... \wedge x_n \preccurlyeq \hat{\phi}$. We present two axioms here. Both follow if the encryption is CCA2 secure and if random generation is only guessable with negligible probability. The first expresses secrecy, the second non-malleability. None of them implies the other. As they are not trivial, we state them in the form of theorems.

▶ **Theorem 4.1** (Secrecy of CCA2 Encryption). *If the encryption scheme is IND-CCA2, then the following formula is computationally sound.*

$$\mathsf{RandGen}(K) \ \wedge \ eK \sqsubseteq \hat{\phi} \ \wedge \ \mathsf{fresh}(R; \hat{\phi}, \vec{x}, x, y) \ \wedge \ \vec{x}, x, y \preccurlyeq \hat{\phi} \ \wedge \ \hat{\phi}, \vec{x}, \{x\}^R_{eK} \vdash y$$
$$\longrightarrow \ dK \sqsubseteq_{\not\emptyset} \hat{\phi}, \vec{x}, x \ \vee \ \hat{\phi}, \vec{x} \vdash y$$

This axiom is a stronger version of the one proved in [7] and its proof is available in the full version of our paper [5]. It says that if K was correctly generated, R is fresh, and y can be derived with the help of $\{x\}^R_{eK}$, then it can be derived without $\{x\}^R_{eK}$, or dK has been sent out or it is in $\vec{x}$ or x. The condition $dK \not\sqsubseteq_{\not\emptyset} \hat{\phi}, \vec{x}, x$ (if moved to the premise) ensures dK has not been revealed, and it is also an easy way to avoid key-cycles, sufficient for the NSL protocol. A tighter axiom is left for future work. Again, $\vec{x}, x, y \preccurlyeq \hat{\phi}$ ensures that $\vec{x}, x, y$ do not contain future information in the form of handles not computable from ϕ. Note, that this axiom may look like it would work for CPA security, but it does not in general, as in general honest agents can be used as decryption oracles. For proving this axiom we need the decryption oracle in the CCA2 game *before* the ciphertext was created by the encryption oracle, but not after.

Let us now consider the case of non-malleability and suppose that we have pairing as before. Let $f_1, ..., f_n$ be the rest of the non-0-arity function symbols not in $\mathcal{F}_c \cup \mathcal{F}_d$ from Section 3. Let $\mathsf{maycorrupt}_{\mathrm{CCA2}}(u; \hat{\phi}, \vec{x})$ be a constraint meaning that u is a term that is paired-together all terms which occur in $\phi, \vec{x}$ not guarded by an honest encryption, immediately under one of the functions $f_1, \ldots, f_n$ or immediately under an honest decryption, not in the key position.

▶ **Theorem 4.2** (Non-Malleability of CCA2 Encryption). *If the encryption scheme is IND-CCA2, then the following formula is computationally sound.*

$$\exists u(\mathsf{maycorrupt}_{\mathrm{CCA2}}(u;\hat{\phi},\vec{x})\wedge\hat{\phi},\vec{x},u\not\vdash N)\ \wedge\ \mathsf{RandGen}(N)\ \wedge\ \mathsf{RandGen}(K)$$
$$\wedge\ eK\sqsubseteq\hat{\phi}\ \wedge\ \vec{x}\preccurlyeq\hat{\phi}\ \wedge\ N\sqsubseteq\hat{\phi},\vec{x}\ \wedge\ \hat{\phi},\vec{x}\vdash y\ \wedge\ \hat{\phi},\vec{x},dec(y,dK)\vdash N$$
$$\rightarrow\ \exists K'(\mathsf{RandGen}(K')\wedge dK'\sqsubseteq_{\not d}\hat{\phi},\vec{x})\vee\hat{\phi},\vec{x}\vdash N\vee\exists xR(y=\{x\}^R_{eK}\wedge\{x\}^R_{eK}\sqsubseteq\hat{\phi},\vec{x})$$

This means that if N and K were correctly generated, y can be decrypted and from the plaintext, N can be derived, but no honest agent ever produced y as an encryption, then either N can be derived without the plaintext of y, or some dK' has been sent out. For this, we need the full power of the CCA2 security, decryption oracle calls both before and after encryption oracle calls. The first conjunct is necessary for making sure that function symbols other than the ones related to pairing or encryption do not interfere with our CCA2 encryption. In principle it is possible to have another encryption for example that may allow to fake encryptions of our CCA2 encryption. Also, there is no guarantee for a CCA2 encryption that, for example, $dec(N, dK)$ does not decrypt to N. In the NSL case, $\mathsf{maycorrupt}_{\mathrm{CCA2}}$ only refers to decryptions as we do not have other function symbols. The proof is in [5].

- Special to c_i, c_r. These axioms are trivial as c_i, c_r are just ideal functions introduced for convenience to represent the agents' view of a session. (Let c be either of them):
  - c does not help the adversary: $\mathsf{RandGen}(N)\wedge\hat{\phi},\vec{x},c(x,y,z,w)\vdash N\rightarrow\hat{\phi},\vec{x}\vdash N$
  - c cannot be forged and cannot be subpart of a term:
    $\hat{\phi},\vec{x}\vdash c(x,y,z,w)\longrightarrow c(x,y,z,w)\sqsubseteq\hat{\phi}\vee x_1=c(x,y,z,w)\vee\ldots\vee x_l=c(x,y,z,w)$
  - c cannot be equal to anything else: If the outermost function symbol of a term T is something different from c, then $c(x,y,z,w)\neq T$.
- **One Extra Axiom** For the NSL protocol, we need an additional axiom, namely,
  - $\mathsf{RandGen}(N)\rightarrow\neg W(\pi_2(N))$.

  That is, the second projection of a nonce can never be a name (by overwhelming probability on a non-negligible set). We assume that the implementation of the pairing is such that this condition is satisfied. If this does not hold, there is an attack which we include in Section 5. It is very easy to ensure that an implementation satisfies this property. If the length of nonces is fixed for a given security parameter, and agents check the length of bit strings that are supposed to be nonces, in this case $\pi_1(N)$, then we can prevent $W(\pi_2(N))$ as it is easy to show that $W(\pi_2(N))$ is only possible if the length of $\pi_1(N)$ differs from the length of N with non-negligible probability. This means that *security of the NSL protocol does not require tagging of nonces, pairs, encryptions.* This axiom is used in step in 2.2.2 of the full proof of the NSL protocol presented in [5].

5 An Attack on NSL

If we assume that $\mathsf{RandGen}(N)\wedge W(\pi_2(N))$ is computationally satisfiable, then we have the following computational attack on the NSL protocol. $\mathsf{RandGen}(N)\wedge W(\pi_2(N))$ is the same as saying that with non-negligible probability, it is possible to choose a name (bit string) Q for an agent such that for the output N of some honest nonce generation, there is a bit string n such that $\langle n, Q\rangle = N$. To show that this is not at all unrealistic, suppose that the pairing $\langle\cdot,\cdot\rangle$ is concatenation, and the length of agent names does not depend on the security parameter, say always 8 bits. Then for any name Q, n can be chosen with $\langle n, Q\rangle = N$ as long as the last four digits of N equals Q, which, if N is evenly generated, is of just $1/2^8$, i.e. non-negligible probability. So this situation is realistic. Now, the attack is the following, it needs two sessions:

1. The adversary choses a name Q as above.

2. The adversary catches the last message $\{N_2\}_B$ in a session between A and B, two honest agents.
3. The adversary, acting as agent Q initiates a new session with B, sending $\{N_2\}_B$ to him.
4. Since B believes this is a new session with Q, it will parse $\{N_2\}_B$ according to its role, namely as $\{N_1', Q\}_B$. This will succeed as long as there is an n with $\langle n, Q\rangle = N_2$, that is, it will succeed with non-negligible probability $1/2^8$.
5. B then generates a new nonce, N_2', and sends $\{n, N_2', B\}_Q$ to Q.
6. The adversary Q decrypts $\{n, N_2', B\}_Q$, reads n, and computes $N_2 = \langle n, Q\rangle$. The secrecy of N_2 is hence broken.

So, we can conclude that if $\langle n, Q\rangle = N$ is possible computationally with non-negligible probability, then the protocol fails. In such case, trace-lifting soundness proofs fail as a bit string can be understood both as $\langle n, Q\rangle$ and as N.

Clearly, if the implementation of the protocol is such that B always checks the length of n, then this attack is not possible. It just has to be made sure somehow that the implementation satisfies the $\mathsf{RandGen}(N) \to \neg W(\pi_2(N))$ property.

Notice, that this attack is not a usual type-flaw attack, because even if type-flaw attacks are allowed, honestly generated nonces are normally considered atomic. For example, the reader may suggest that this attack is in fact very similar to the one shown in [19] (as we both wrote it as $N = \langle n, Q\rangle$). However, there is a very fundamental difference. The attack in [19] is based on the fact that an honest agent sends a pair with a nonce and an agent name, and the receiving honest agent understands this as a single nonce. In other words, in [19] the honest receiver reads the pair of a nonce and a name into an input variable meant for a nonce. There, n is the honest nonce and N is the input variable. In our attack, it is an actual nonce that is understood by the receiver as a pair of a nonce and a name. In our case, an actual nonce is read into the pair of two input variables: one for a nonce and another for a name. Here N is the honest nonce and n corresponds to the input variable. This is a fundamental difference as in our case there are no atomic objects at all. Even an honest nonce is allowed to be split. To our knowledge, this is the first such attack on the NSL protocol.

6 Examples for Proving Inconsistency

Here we look at three small example proofs so that the reader can become familiar with how the axioms work. Note that these derivations are *not pure first-order deductions*. Not only we use the axioms and first order deduction rules, but *we also use how a symbolic execution is defined*.

▶ **Example 6.1.** We start with a very trivial example. It is rather obvious that in the execution of the NSL protocol in Example 3.1, $\phi_2 \not\vdash A$ should be inconsistent with the axioms as A is included in ϕ_2. We can derive it the following way: observe that

$$\phi_2 \equiv A, B, eK_A, eK_B, \{\langle N_1, A\rangle\}^{R_1}_{eK_B} \equiv \phi_0, A, B, eK_A, eK_B, \{\langle N_1, A\rangle\}^{R_1}_{eK_B}. \tag{1}$$

From the self-derivability axiom at step 0, $\phi_0, B, eK_A, eK_B, \{\langle N_1, A\rangle\}^{R_1}_{eK_B}, A \vdash A$. By commutativity, $\phi_0, A, B, eK_A, eK_B, \{\langle N_1, A\rangle\}^{R_1}_{eK_B} \vdash A$, so its negation is inconsistent with the axioms. Hence, $\mathcal{M}, \sigma \not\models A, B, eK_A, eK_B, \{\langle N_1, A\rangle\}^{R_1}_{eK_B} \not\vdash A$ for any model $\mathcal{M}$, which implies by (1) that $\phi_2 \not\vdash A$ is inconsistent with the axioms.

▶ **Example 6.2.** We can also derive, as expected, that $\phi_2 \vdash N_1$ is inconsistent with the axioms in our NSL example. This should be the case, as N_1 has only been sent out under

a single good encryption. As $\phi_2 \equiv A, B, eK_A, eK_B, \{\langle N_1, A\rangle\}^{R_1}_{eK_B} \equiv \phi_1, \{\langle N_1, A\rangle\}^{R_1}_{eK_B}$, it is enough to show that $\phi_1, \{\langle N_1, A\rangle\}^{R_1}_{eK_B} \vdash N_1$ is inconsistent with the axioms. Suppose, in order to get a contradiction, that this is not the case, *i.e.*, $\phi_1, \{\langle N_1, A\rangle\}^{R_1}_{eK_B} \vdash N_1$. To apply the secrecy axiom consider that $\vec{x} = \langle\rangle, x = \langle N_1, A\rangle$, and $y = N_1$. Since K_B was correctly generated (appeared as a name), $\mathsf{RandGen}(K_B)$ holds. By Example 3.1 we have $eK_B \sqsubseteq \phi_1$. $\mathsf{fresh}(R_1; \phi_1, \vec{x}, x, y)$ also holds because $\mathsf{RandGen}(R_1) \wedge R_1 \not\sqsubseteq \phi_1, \langle\rangle, \langle N_1, A\rangle, N_1$. Finally, since $\vec{x} \preccurlyeq \phi_1$, $x \preccurlyeq \phi_1$ and $y \preccurlyeq \phi_1$ because none has handles, $\phi_1, \{\langle N_1, A\rangle\}^{R_1}_{eK_B} \vdash N_1$ was supposed, and $dK_B \not\sqsubseteq \phi_1, \vec{x}, x$, one may apply the secrecy axiom and get $\phi_1 \vdash N_1$. At Step 1 we have $\mathsf{fresh}(N_1; \phi_1)$ as $\mathsf{RandGen}(N_1) \wedge N_1 \not\sqsubseteq \phi_1$, so $\mathsf{fresh}(N_1; \phi_1) \wedge \phi_1 \vdash N_1$ holds, which contradicts the no telepathy axiom.

▶ **Example 6.3.** From the axioms, we can also derive the increasing knowledge of an adversary, *i.e.*, for any m and x, if $\phi_m \vdash x$ is derivable from the axioms and agent checks, then $\phi_{m+1} \vdash x$ is also derivable from the axioms and agent checks. Assume that $\phi_m \vdash x$ is derivable from the axioms and agent checks. Let t be the message sent in the $m+1$'th step *i.e.*, $\phi_{m+1} \equiv \phi_m, t$. The increasing capabilities axiom applied to step m means $\phi_m \vdash x$ implies $\phi_m, t \vdash x$. But that is the same as $\phi_{m+1} \vdash x$. Note that from the above and from Example 6.1 it also follows that for any m, the axioms imply that $\phi_m \vdash A$: it is clear from Example 6.1, that $\phi_2 \vdash A$ is implied, then from the above, by induction, $\phi_m \vdash A$ is also implied. Note that the induction is not within FOL, we used the induction on the number of execution steps.

7 Correctness Proof of NSL

We present now the correctness of the NSL protocol for any (bounded) number of sessions. We show that in the symbolic execution defined above, violation of secrecy or authentication is inconsistent with the axioms. As we mentioned, we assume that agent A only executes the initiator role, and agent B only executes the responder role. But, we allow both A and B to have other sessions running with possibly corrupted agents. We start by showing that nonces that were generated by honest initiator A and sent to honest responder B, or vice-versa, remain secret. We do this by picking any step m of the execution tree, and listing all possible messages sent by A and B, show that $\phi_m \not\vdash N$ together with the axioms and agent checks imply $\phi_{m+1} \not\vdash N$ for each possible sent message. Hence, $\phi_m \not\vdash N$, the axioms and the agent checks, and $\phi_{m+1} \vdash N$ are inconsistent. Since $\phi_0 \not\vdash N$ initially holds by no-telepathy, by induction we have $\phi_m \not\vdash N$ after any finite number of steps m. The reader can see below that the induction hypothesis is a little more complex, but essentially this is what we do.

Once secrecy is proven, authentication and agreement are shown. We pick the point on the execution tree when the responder finished his task, and using that nonces remain secret, together with non-malleability, we show that the initiator also finished his task and the corresponding values seen by the two parties have to match. In other words, B finished, A not finished or values don't match, and the axioms and the agent checks are inconsistent.

7.1 Secrecy

The aim of the secrecy proof is to show that nonces N sent between A and B remain secret. The fact that N is a nonce sent by A to B in the NSL protocol can be expressed as $\exists R\big(\{N, A\}^R_{eK_B} \sqsubseteq \hat{\phi}\big)$. If B sent it to A, that means $\exists hR\big(\{\pi_1\left(dec(h, dK_B)\right), N, B\}^R_{eK_A} \sqsubseteq \hat{\phi}\big)$. So, such nonces can be characterized by the condition

$$C[N] \equiv \mathsf{RandGen}(N) \wedge \big(\exists R.\{N, A\}^R_{eK_B} \sqsubseteq \hat{\phi} \vee \exists hR.\{\pi_1\left(dec(h, dK_B)\right), N, B\}^R_{eK_A} \sqsubseteq \hat{\phi}\big).$$

Then the secrecy property we want to show is that $\forall N\big(C[N] \longrightarrow \hat{\phi} \not\vdash N\big)$, meaning that such nonces cannot be computed by the adversary. It is equivalent to show that its negation, $\exists N\big(C[N] \wedge \hat{\phi} \vdash N\big)$, is inconsistent with the axioms and the agent checks on every possible symbolic trace.

Suppose the total length of the symbolic trace in question is n. At the end of the trace the frame ϕ contains n terms. Let us denote the frames at each node of this trace by ϕ_0, ϕ_1, ϕ_2, etc. Each frame contains one more term than the previous one. Satisfaction of $C[N]$ by this trace means that one of the terms $\{N, A\}^R_{eK_B}$ or $\{\pi_1\left(dec(h, dK_B)\right), N, B\}^R_{eK_A}$ appears in frame ϕ_n for some h, R. Let us fix such N. If $\vec{x}$ is a list of a finite number of nonces $\vec{x} \equiv N_1, ..., N_l$ that were all generated by either A or B (possibly intended to each other, possibly intended for other possibly malicious agents), and they are all different from N, then we say condition $C'[\vec{x}, N]$ is satisfied:

$$C'[N_1, ..., N_l, N] \equiv \bigwedge_{i=1}^{l} \left(\begin{array}{l} \mathsf{RandGen}(N_i) \ \wedge \ N \neq N_i \ \wedge \ \big(\exists QR.\{N_i, A\}^R_{eK_Q} \sqsubseteq \hat{\phi} \vee \\ \exists QhR.\{\pi_1\left(dec(h, dK_B)\right), N_i, B\}^R_{eK_Q} \sqsubseteq \hat{\phi}\big) \end{array} \right)$$

We will carry out an inductive proof on the length of ϕ_n. As it turns out, in order to avoid loops in the proof, instead of $\exists N\big(C[N] \wedge \hat{\phi} \vdash N\big)$, it is better to prove that

$$\exists N \exists \vec{x}\big(C[N] \wedge C'[\vec{x}, N] \wedge \hat{\phi}, \vec{x} \vdash N\big) \tag{2}$$

is inconsistent with the axioms and agent checks. On the symbolic trace, this means that for all n,

$$\exists N \exists \vec{x}\big(C[N] \wedge C'[\vec{x}, N] \wedge \phi_n, \vec{x} \vdash N\big)$$

is inconsistent with the axioms and agent checks. We do this by fixing an arbitrary N satisfying $C[N]$, and by showing that if for some $m < n$, $\exists \vec{x}\big(C'[\vec{x}, N] \wedge \phi_m, \vec{x} \vdash N\big)$ is inconsistent with the axioms and agent checks, then $\exists \vec{x}\big(C'[\vec{x}, N] \wedge \phi_{m+1}, \vec{x} \vdash N\big)$ is also inconsistent with the axioms and agent checks. As at $m = 0$ the statement follows from no telepathy, we are done. To be completely precise, we would have to take into account $\mathsf{maycorrupt}_{\mathrm{CCA2}}$ as well, but this is trivial for NSL as only those things are decrypted that came from the clear (only handles are decrypted). Again, note that within C and C', $\hat{\phi}$ is always ϕ_n and not ϕ_m.

▶ **Proposition 7.1.** *In the above execution of NSL protocol, let N be such that $C[N]$ is satisfied, and let $m < n$. If for all $\vec{x}$ such that $C'[\vec{x}, N]$ holds, the axioms and agent checks imply (by FOL deduction rules) that $\phi_m, \vec{x} \not\vdash N$, then for all $\vec{x}$ such that $C'[\vec{x}, N]$ holds, the axioms and agent checks imply (by FOL deduction rules) that $\phi_{m+1}, \vec{x} \not\vdash N$ holds.*

The proof is in [5]. Once this is shown, we still have to prove that the property initially holds, that is, $\exists N \exists \vec{x}\big(C[N] \wedge C'[\vec{x}, N] \wedge \phi_0, \vec{x} \vdash N\big)$ is inconsistent with the axioms. Let $C[N]$ and $C'[\vec{x}, N]$ hold for N and $\vec{x} \equiv N_1, ..., N_l$. At step 0, N, $N_1, ..., N_l$ are still fresh (remember, we assumed for simplicity that everything was generated upfront, and clearly, these nonces have not been sent), so by the no telepathy axiom, $\phi_0 \not\vdash N$, and then by the independence of fresh items, $\phi_0, N_1 \not\vdash N$. Then again by the independence of fresh items, $\phi_0, N_1, N_2 \not\vdash N$, etc. So $\phi_0, N_1, ..., N_l \not\vdash N$ holds, meaning that $\exists N \exists \vec{x}\big(C[N] \wedge C'[\vec{x}, N] \wedge \phi_0, \vec{x} \vdash N\big)$ is indeed inconsistent with the axioms. Therefore, together with the induction step of Proposition 7.1, we have:

▶ **Theorem 7.2** (Secrecy). *Consider a symbolic execution of the NSL protocol, with an arbitrary number of possible dishonest participants and two honest participants A, B that follow the initiator and responder roles correspondingly, and that only execute these roles*

in each of their bounded number of sessions. Further, consider the convention $\langle x, y, z\rangle \equiv \langle x, \langle y, z\rangle\rangle$. *Our axioms together with the agent checks and* $\mathsf{RandGen}(N) \rightarrow \neg W(\pi_2(N))$ *imply that for any* $n \in \mathbb{N}$ *and for any nonce* N *that was either generated by* A *and sent to* B*, or vice versa,* $\phi_n \not\vdash N$.

The above Theorem states that secrecy of nonces satisfying $C[N]$ is never broken. That is, nonces that were generated by A or B and intended to be sent between each other, remain secret. In particular, asking $\vec{x}$ to be the empty list, the formula $\exists N\big(C[N] \wedge \hat{\phi} \vdash N\big)$, together with the axioms and the agent checks, and $\mathsf{RandGen}(N) \rightarrow \neg W(\pi_2(N))$ are inconsistent on any symbolic trace.

7.2 Agreement and Authentication

We now prove agreement from the responder's viewpoint. That is, we will show that

$$\begin{array}{l} Resp^B_{NSL}[B, i', N_2, h_2, h_4, R_2] \quad \text{AND} \\ \pi_2(dec(h_2, dK_B)) = A \end{array} \quad \Longrightarrow \quad \begin{array}{l} \text{EXIST} \;\; i, N_1, h_1, h_3, R_1, R_3 \;\; \text{SUCH THAT} \\ Init^A_{NSL}[A, i, B, N_1, h_1, h_3, R_1, R_3] \;\; \text{AND} \\ dec(h_2, dK_B) = \langle N_1, A\rangle \;\; \text{AND} \\ dec(h_3, dK_A) = \langle N_1, N_2, B\rangle \;\; \text{AND} \\ dec(h_4, dK_B) = N_2 \end{array}$$

where by the implication sign we mean that the agent checks and the axioms imply this. We can also write this within our syntax:

$$\begin{array}{l} A = \pi_2(dec(h_2, dK_B)) \;\wedge \\ N_1 = \pi_1(dec(h_2, dK_B)) \;\wedge \\ c_r(A, B, N_1, N_2) \sqsubseteq \hat{\phi} \;\wedge \end{array} \longrightarrow \exists h_3. \left(\begin{array}{l} c_i(A, B, N_1, N_2) \sqsubseteq \hat{\phi} \;\wedge \\ N_2 = \pi_1(\pi_2(dec(h_3, dK_A))) \end{array} \right)$$

What we have to prove is that the negation of this is inconsistent with the axioms and agent checks. But for that it is sufficient to show that the agent checks and axioms, and the premise of the formula imply the conclusion of this formula, as the following theorem states with the proof available in [5].

▶ **Theorem 7.3** (Agreement and Authentication). *Consider a symbolic execution of the NSL protocol, with an arbitrary number of possible dishonest participants and two honest participants* A*,* B *that follow the initiator and responder roles correspondingly, and that only execute these roles in each of their bounded number of sessions. Further, consider the convention* $\langle x, y, z\rangle \equiv \langle x, \langle y, z\rangle\rangle$.

Our axioms together with the agent checks and $\mathsf{RandGen}(N) \rightarrow \neg W(\pi_2(N))$ *are inconsistent with the negation of the formula*

$$\begin{array}{l} c_r(\pi_2(dec(h_2, dK_B)), B, \pi_1(dec(h_2, dK_B)), N_2) \sqsubseteq \hat{\phi} \;\wedge\; A = \pi_2(dec(h_2, dK_B)) \\ \longrightarrow \;\; \exists N_1 h_3. \left(\begin{array}{l} c_i(A, B, N_1, \pi_1(\pi_2(dec(h_3, dK_A)))) \sqsubseteq \hat{\phi} \;\wedge \\ N_2 = \pi_1(\pi_2(dec(h_3, dK_A))) \;\wedge\; N_1 = \pi_1(dec(h_2, dK_B)) \end{array} \right) \end{array}$$

8 Conclusion and Future Work

In this paper we illustrated that the framework proposed by Bana and Comon-Lundh [7], where one does not define explicitly the Dolev-Yao adversarial capabilities but rather the limitations (axioms) on these capabilities, is suitable and powerful enough to prove correctness of security protocols. The proofs with this technique are computationally sound without the need of any further assumptions such as no bad keys, etc that are otherwise usually assumed in other literature.

We presented a modular set of axioms that are computationally sound for implementations using CCA2 secure encryption. Using the axioms together with a minimal parsing assumption, we were able to perform an inductive proof to show both secrecy and agreement of the NSL protocol. Applying the main theorem of [7] we obtain that for any implementation satisfying CCA2 security and the parsing assumption, there is no computational adversary that can violate secrecy or authentication except with negligible probability. We also believe the axioms of secrecy and non-malleability constitute a sufficient abstraction of CCA2 security to prove correctness of protocols other than NSL.

As other current techniques have problems incorporating dynamic corruption, it is worth noting, that our technique works even if the protocol allows the release of a decryption key of A or B at some time. Secrecy can still be proven until that point, and authentication that was carried out earlier can be verified even if the decryption key is later released.

The proof we presented in this paper was done by hand but we believe that automation is possible and is left for future work.

References

1. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, January 2002.
2. P. Adão, G. Bana, J. Herzog, and A. Scedrov. Soundness and completeness of formal encryption: the cases of key-cycles and partial information leakage. *Journal of Computer Security*, 17(5):737–797, 2009.
3. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *CCS'03*, pages 220–230. ACM, 2003.
4. M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.
5. G. Bana, P. Adão, and H. Sakurada. Computationally Complete Symbolic Attacker in Action—Long version, 2012. Available at `http://eprint.iacr.org/2012/316`.
6. G. Bana and H. Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker, 2012. Available at `http://eprint.iacr.org/2012/019`.
7. G. Bana and H. Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In *POST'12*, volume 7215 of *LNCS*, pages 189–208. Springer, 2012.
8. G. Bana, K. Hasebe, and M. Okada. Computational Semantics for First-Order Logical Analysis of Cryptographic Protocols. In *Formal to Practical Security*, volume 5458 of *LNCS*, pages 33–56. Springer, 2007.
9. G. Bana, K. Hasebe, and M. Okada. Secrecy-oriented first-order logical analysis of cryptographic protocols, 2010. Available at `http://eprint.iacr.org/2010/080`.
10. G. Bana, P. Mohassel, and T. Stegers. Computational Soundness of Formal Indistinguishability and Static Equivalence. In *ASIAN'06*, volume 4435 of *LNCS*, pages 182–196. Springer, 2007.
11. G. Barthe, B. Grégoire, and S. Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL'09*, pages 90–101. ACM, 2009.
12. M. Baudet, V. Cortier, and S. Kremer Computationally Sound Implementations of Equational Theories Against Passive Adversaries. In *ICALP'05*, volume 3580 of *LNCS*, pages 652–663. 2005.
13. B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, 2008.
14. H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *CCS'08*, pages 109–118. ACM, 2008.

15 H. Comon-Lundh and V. Cortier. How to prove security of communication protocols? A discussion on the soundness of formal models w.r.t. computational ones. In *STACS'11*, volume 9 of *LIPIcs*, pages 29–44. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

16 V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *ESOP'05*, volume 3444 of *LNCS*, pages 157–171. Springer, 2005.

17 V. Cortier and B. Warinschi. A composable computational soundness notion. In *CCS'11*, pages 63–74. ACM, 2011.

18 A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *ICALP'05*, volume 3580 of *LNCS*, pages 16–29. 2005.

19 J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS'01*, pages 166–175. ACM, 2001.

20 B. Warinschi. A computational analysis of the Needham-Schroeder protocol. In *CSFW'03*, pages 248–262. IEEE Computer Society, 2003.