39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2019, December 11–13, 2019, Bombay, India

Edited by

Arkadev Chattopadhyay
Paul Gastin

LIPICS

Editors

Arkadev Chattopadhyay
Tata Institute of Fundamental Research, Mumbai, India
arkadev.c@tifr.res.in

Paul Gastin
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
Paul.Gastin@ens-paris-saclay.fr

ACM Classification 2012
Theory of computation

ISBN 978-3-95977-131-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-131-3.

Publication date
December, 2019

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

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2019.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019) was held at the Indian Institute of Technology Bombay, India from December 11 to December 13, 2019. In this version, a new format was introduced as the conference was formally separated into two tracks, Track-A focussing on algorithms, complexity and related issues and Track-B focussing on logic, automata and other formal methods aspects of computer science. Each track had its own Program Committee (PC) with a chair. This volume constitutes the joint proceedings of the two tracks, published in the LIPIcs series under a Creative Common license, with free online access to all.

The conference comprised of 6 invited talks, 22 contributed talks in Track-A and 23 in Track-B. This volume contains all contributed papers from both tracks and the abstracts of all invited talks presented at the conference. There were overall 93 submissions (after weeding out the obvious bogus ones), 51 in Track-A and 42 in Track-B. We thank all those who submitted to FSTTCS. We also thank all PC members for their tireless work and all external reviewers for their expert opinion in the form of timely reviews.

We are grateful to all the invited speakers: Karthikeyan Bhargavan (INRIA-Paris, France), Robert Krauthgamer (Weizmann, Israel), Ranko Lazić (Warwick, U.K.), Toniann Pitassi (Toronto, Canada), Tim Roughgarden (Columbia, U.S.) and Alexandra Silva (University College London, U.K.). They kindly accepted our invitations and gave talks that inspired the entire FSTTCS audience at large.

The main conference was preceded by two workshops on the 10th December: *Complexity in Algorithmic Game Theory*, organized by Siddharth Barman (IISc.) and Umang Bhaskar (TIFR), and *Trends in Transformations*, organized by S. Krishna (IIT Bombay). This was followed by two post-conference workshops on 14th December: *Extension Complexity and Lifting Theorems*, organized by Ankit Garg (MSR Bangalore), Raghu Meka (UCLA), Toniann Pitassi (Toronto) and Makrand Sinha (CWI, Amsterdam), and *GALA: Gems of Automata, Logic and Algebra*, organized by C. Aiswarya (CMI, Chennai), S. Akshay (IIT Bombay) and Benedikt Bollig (LSV, CNRS & ENS Paris-Saclay). Besides, there was a colocated SAT/SMT winter school from December 8th to 10th.

We are indebted to the organizing committee members from the Department of Computer Science and Engineering, IIT Bombay for ensuring a smooth running of the conference and workshops and making all necessary arrangements. We also thank S.P. Suresh of CMI for maintaining and working on the conference web page. We also thank the friendly staff at Dagstuhl LIPICs, particularly Michael Wagner, for being prompt and helpful in answering our queries. Finally, we would also like to thank the members of the Steering Committee for having faith in us for running the conference.

Arkadev Chattopadhyay and Paul Gastin
December 2019

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Reviewers

Track-A

Saba Ahmadi
Josh Alman
Vishwas Bhargava
Anup Bhattacharya
Sayan Bhattacharya
Ivan Bliznets
Joshua Brakensiek
Deeparnab Chakrabarty
Sourav Chakraborty
Arkadev Chattopadhyay
Bhaskar Ray Chaudhury
Xi Chen
Suryajith Chillara
Rajesh Chitnis
Chi-Ning Chou
Marek Cygan
Prem Laxman Das
Syamantak Das
Dariusz Dereniowski
Christian Engels
Yuval Filmus
Kshitij Gajjar
Francois Le Gall
Ankit Garg
Jugal Garg
Alex Gavryushkin
Parikshit Gopalan
Rupert Hölzl
Hossein Jowhari
Sagar Kale
Pieter Kleer
Amit Kumar
Mrinal Kumar
Gad Landau
Bruno Loff
Satya Lokam
Brendan Lucier
Diptapriyo Majumdar
Yury Makarychev
Nikhil Mande
Barnaby Martin
Andrew McGregor
Klaus Meer
Or Meir
Neeldhara Misra
Pranabendu Misra
Rajat Mittal
Anish Mukherjee
Maryam Negahbani
Patrick K. Nicholson
Prajakta Nimbhorkar
Ioannis Panageas
Geevarghese Philip
Aditya Potukuchi
David Purser
Jaikumar Radhakrishnan
Sridharan Ramanujan
Atri Rudra
Barna Saha
Chandan Saha
Rahul Santhanam
Ramprasad Saptharishi
Jayalal Sarma
Kanthi Sarpatwar
Nitin Saurabh
Lena Schlipf
C. Seshadhri
Suhail Sherif
Anastasios Sidiropoulos
Avishay Tal
Kavitha Telikepalli
Roei Tell
Meng-Tsung Tsai
Rahul Vaze
Ben Lee Volk
Thomas Watson
Sheng Yang
Yu Yokoi
Chihao Zhang

Track-B

Parosh Aziz Abdulla
Sergio Abriola
C. Aiswarya
S. Akshay
Clément Aubert
Nikolaj Bjorner
Michael Blondin
Udi Boker
Benedikt Bollig
Florian Bruse
Véronique Bruyère
Michaël Cadilhac
Simon Castellan
Debraj Chakraborty
Vincent Cheval
Dmitry Chistikov
Pierre Clairambault
Wojciech Czerwiński
Deepak D'Souza
Dana Drachsler Cohen
Mahsa Eftekhari Hesari
Ehab Elsalamouny
Mohamed Faouzi Atig
Hongfei Fu
Pierre Ganty
Paul Gastin
David Gross-Amblard
Shibashis Guha
Christoph Haase
Hsi-Ming Ho
Markus Holzer
Suresh Jagannathan
Jean-Pierre Jouannaud
Marcin Jurdzinski
Joost-Pieter Katoen
Christoforos Keroglou
Stefan Kiefer
Alexander Knapp
Alexander Knop
Steve Kremer
S. Krishna
Anna Labella
Jean-Jacques Levy
Kamal Lodaya
Christof Löding
Amaldev Manuel
Nicolas Markey
Bastien Maubert
Filip Mazowiecki
Alexander Meduna
Lukasz Mikulski
Benjamin Monmege
Antoine Mottet
Madhavan Mukund
Maurizio Murgia
K. Narayan Kumar
Uwe Nestmann
Youssouf Oualhadj
Catuscia Palamidessi
Erik Paul
Vincent Penelle
Guillermo Perez
Sophie Pinchinat
Anton Pirogov
Thomas Place
Sanjiva Prasad
M. Praveen
Karin Quaas
Raghavan Komondoor
R. Ramanujam
Narad Rampersad
Jean-Francois Raskin
Prakash Saivasan
Sven Schewe
Sylvain Schmitz
Stefan Schwoon
Salomon Sickert
Michał Skrzypczak
Mate Soos
Sriram Sankaranarayanan
B. Srivathsan
S. P. Suresh
Ramanathan S. Thinniyam
Rob van Glabbeek
Vincent Van Oostrom
Sarah Winter
Georg Zetzsche
Martin Zimmermann

Practical Formal Methods for Real World Cryptography

Karthikeyan Bhargavan
Inria, Paris, France
karthikeyan.bhargavan@inria.fr

Prasad Naldurg
Inria, Paris, France
prasad.naldurg@inria.fr

Abstract

Cryptographic algorithms, protocols, and applications are difficult to implement correctly, and errors and vulnerabilities in their code can remain undiscovered for long periods before they are exploited. Even highly-regarded cryptographic libraries suffer from bugs like buffer overruns, incorrect numerical computations, and timing side-channels, which can lead to the exposure of sensitive data and long-term secrets. We describe a tool chain and framework based on the F∗ programming language to formally specify, verify and compile high-performance cryptographic software that is secure by design. This tool chain has been used to build a verified cryptographic library called HACL∗, and provably secure implementations of sophisticated secure communication protocols like Signal and TLS. We describe these case studies and conclude with ongoing work on using our framework to build verified implementations of privacy preserving machine learning software.

2012 ACM Subject Classification Security and privacy → Formal security models; Security and privacy → Logic and verification

Keywords and phrases Formal verification, Applied cryptography, Security protocols, Machine learning

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.1

Category Invited Talk

1 Introduction

Cryptography is the backbone of most internet applications, including e-commerce, online payment, messaging, social networking, and user communications. Different algorithms and protocols are used to guarantee different levels of confidentiality, integrity and authentication protection, depending on application and user requirements. In some applications, its use can be opaque to end users, such as in digital rights management and business analytics. While there is no need to motivate the use of cryptography online, implementing cryptographic software for real world applications can be incredibly complex and error-prone. Though governments, companies, and standards bodies have been using and stress-testing cryptographic algorithms for more than twenty years, surprisingly, there is a lack of rigour in how many new protocols and applications are implemented.

Implementations of cryptographic primitives can have obvious as well as subtle vulnerabilities that are often difficult to detect. To illustrate, in OpenSSL, a widely used open-source (and hence open to scrutiny) implementation of common cryptographic algorithms, 16 CVEs (common vulnerability and exposures reports) have been issued since 2017 for vulnerabilities in the core cryptographic functions. These bugs range from incorrect implementations of numerical computations (5), to timing side channel attacks (6), and memory safety issues (5). Such programming errors can often be exploited by a remote attacker to tamper with the cryptographic computation, leading to various degrees of exposure, and invalidating the

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 1; pp. 1:1–1:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

security guarantees the algorithm was designed for in the first place. As a typical example, Brumley et al. [19] show how an arithmetic bug in the implementation of an elliptic curve in OpenSSL can be practically exploited to retrieve a victim's long term private key.

Finding such bugs in large codebases that are focused more on high-performance than high-assurance is not an easy task. Software development practices, from good hygiene and code reviews, to unit-testing and fuzzing, are best-effort and usually incapable of finding subtle vulnerabilities. Rather than attempt to find and fix bugs in an ad hoc manner, our philosophy, in line with a number of recent works [20, 7, 46, 15, 6, 39, 26], is to use formal verification to prove the absence of large classes of vulnerabilities by design.

We use the F∗ programming language and verification framework [43] to build HACL∗, a library of verified cryptographic algorithms in C. Given a published standard specification of a cryptographic primitive, we write verified code in F∗ that is memory safe, functionally correct, and resistant to timing side-channels. This code is then compiled to readable C code that is as performant as hand-written C code in state-of-the-art libraries like OpenSSL. HACL∗ supports most of the algorithms used in modern cryptographic protocols and applications, and is currently being used by the Mozilla Firefox Web browser, the WireGuard VPN, the Tezos Blockchain, and the Microsoft WinQuic protocol stack.

HACL∗ provides a robust basis for building high-assurance cryptographic applications, but the cryptographic library is only one component of the security stack. To protect connections between clients and servers, Web applications rely on standardized protocols like Transport Layer Security (TLS) [42]. For end-to-end secure messaging, WhatsApp and Skype rely on a complex cryptographic protocol called Signal [1]. These protocols invoke a series of cryptographic constructions across multiple messages to achieve sophisticated security goals. The overall security of each protocol depends on subtle invariants, which may be falsified by incorrect designs or buggy implementations. For example, the Triple Handshake attacks on the TLS [13] uncovered a protocol design flaw in the way three TLS sessions can be composed together, resulting in an attack on client authentication that had remained undiscovered for 18 years. The SMACK attacks on TLS libraries [9] found a class of implementation bugs that allowed attacker to completely bypass the security of a large subset of HTTPS connections on the Web. Preventing these kinds of attacks requires careful formal analysis.

We observe that cryptographic primitives are themselves getting more complex, with new post-quantum algorithms and homomorphic encryption constructions currently being standardized and deployed. Applications that use these new constructions, such as electronic voting and privacy preserving machine learning, are even more complicated to specify and analyse than traditional cryptographic protocols. Inevitably, attackers are also getting more sophisticated, and the classic network attacker model needs to be augmented with finer distinctions to catch and fix vulnerabilities.

We argue that the combination of complex protocols, sophisticated security properties, and powerful attackers demands a more rigorous treatment of cryptographic software development. In this paper, we describe how we can apply our verification tool chain across all layers of a secure distributed application, starting with cryptographic algorithms (Section 2), to end-to-end protocols with sophisticated security properties (Section 3), all the way to novel privacy-preserving applications (Section 4). Through these case studies, we show how formal methods can play an important role in building high-assurance cryptographic software.

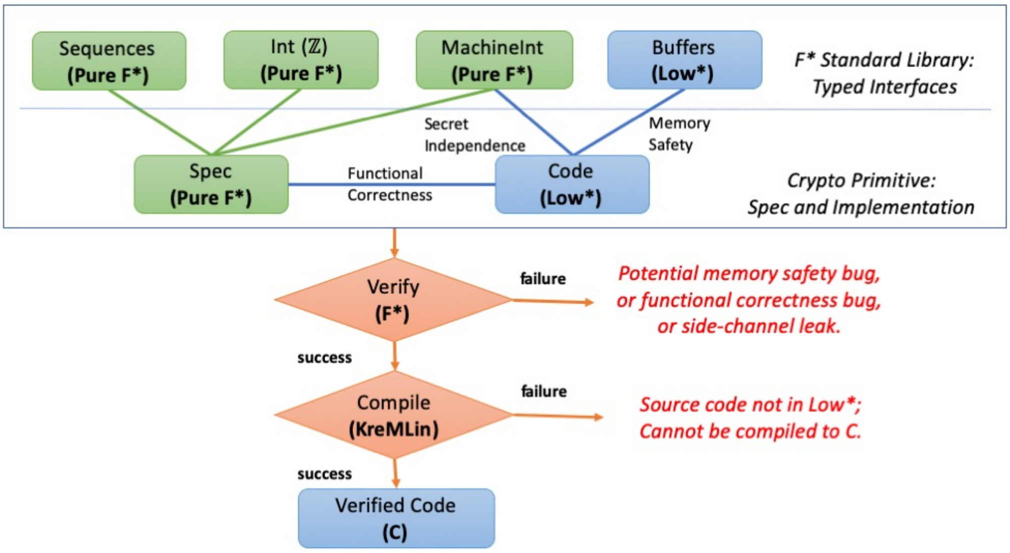


Figure 1 HACL∗ verification and compilation tool chain.

2 Verified Cryptography: HACL∗

HACL∗ [47] is a verified open-source library of modern cryptographic algorithms, including the elliptic curve Curve25519 [3], the authenticated encryption construction ChaCha20-Poly1305 [2], the hash function SHA-2 [44], and the signature scheme Ed25519 [4]. Put together, these algorithms are enough to satisfy all the classic cryptographic needs of a distributed software application. In particular, HACL∗ supports the full NaCl cryptographic API [8], and implements a full ciphersuite of TLS 1.3 [42]. The distributable code of HACL∗ is in portable C, which can be easily wrapped into multiple languages and dropped into application software that needs these algorithms. For example, HACL∗ is currently used to implement TLS in Mozilla Firefox and as the NaCl implementation in the Tezos blockchain.

The verification and compilation tool chain used in the development of HACL∗ is depicted in Figure 1. All the code in HACL∗ is written in F∗, an ML-like functional programming language with a type system that includes polymorphism, dependent types, monadic effects, refinement types, and a weakest precondition calculus [43]. The language is aimed at program verification, and its type system allows the expression of precise and compact functional correctness and security property specifications for programs, which can be mechanically verified, with the help of an SMT solver. After verification, an F∗ program can be compiled to OCaml, F#, C, or even WebAssembly, and so it can run on a variety of platforms.

Figure 1 shows the workflow for adding a new verified cryptographic primitive in HACL∗. The first step is to write a high-level specification (Spec) in a higher-order purely functional subset of F∗. This specification relies on standard libraries for basic datatypes such as mathematical and machine integers ($\mathbb{Z}$,MachineInt), and immutable arrays (Sequences), also written in Pure F∗. Next, an optimized implementation of the primitive itself (Code) is written in Low∗, a low level subset of F∗ that can be efficiently compiled to C, using the KreMLin compiler [41]. For a full description of the syntax, type system, and semantics of F∗, refer to [43], and for the formal development of Low∗ and its compilation to C, see [41].

The Low∗ Code cannot use mathematical integers, and it is only allowed to use machine integer operations in ways that are safe from timing side channels. For example, if an unsigned 32-bit integer (uint32) holds a secret value, e.g. part of an encryption key, it cannot be compared with another integer, it cannot be used as an index into an array, and it cannot be used in a division or modulo operation. This is because, on most hardware platforms, the time taken by these operations may reveal the contents of the secret integer to a remote attacker. Cryptographic code that uses such operations is not *secret independent*, and hence may be vulnerable to various side-channels attacks.

The Low∗ code can also use mutable but memory-safe arrays (Buffers) to hold cryptographic state. However, all the arrays used in HACL∗ are stack-allocated, that is, they never use the heap, and hence do not have to be explicitly allocated or freed.

The code for each crytpographic algorithm is then verified, using the F∗ typechecker, to ensure that it conforms with the logical preconditions and type abstractions in the F∗ library. A failure to type-check here may indicate the presence of memory safety, functional correctness, or side channel vulnerabilities (or that the type checker may need more annotations to prove correctness). If type checking succeeds, the Low∗ code is compiled using KreMLin to portable C code, preserving all the properties verified in F∗.

Surprisingly, writing formally verified cryptographic code in HACL∗ does not have a performance cost. Our C code is as fast as the hand-optimized C code in state-of-the-art cryptographic libraries like OpenSSL. In many cases, the structured compact code generated from F∗ is even faster. Performance is especially important for encryption algorithms and elliptic curves that are used within network protocols like TLS, where cryptography often dominates cost and can be a performance bottleneck. For example, our HACL∗ implementation of Curve25519 was about 20% faster than the previous code for this elliptic curve in Firefox. Hence adopting our code significantly cut the cost of HTTPS connections between Firefox and popular websites like GMail. Similarly, the WireGuard VPN [24], which runs within the Linux Kernel and needs high-performance high-assurance code for Curve25519, uses HACL∗.

HACL∗ is an evolving project. We are extending it with more elliptic curves, encryption algorithms, and hash functions, and use it as a basis for building implementations of more advanced and experimental cryptographic constructions including post-quantum cryptography and homomorphic encryption. As a part of our privacy preserving machine learning project, which we describe in Section 4, we are building verified implementations of several partially homomorphic encryption schemes including Goldwassser-Micali [29], the Paillier [36] additive homomorphic system, and the DGK system [22, 23], using the BigNum library and other verified primitives from HACL∗. To further improve the performance of HACL∗ code, we are building a cryptographic provider called EverCrypt that combines verified C code from HACL∗ with verified assembly code from the Vale project [15].

3 Verified Protocols: LibSignal∗

In this section, we describe how to extend the scope of our security guarantees from cryptographic libraries to cryptographic protocols. Protocols that are built using verified primitives are not automatically secure, and require a different set of tools for specification and verification of higher-level properties.

We illustrate this with our work on Signal, an end-to-end encryption protocol for instant messaging that is used in many popular applications like WhatsApp, Skype, and Facebook Messenger, by billions of users worldwide. The main design goal of Signal is to maximally protect the privacy of its users, even if the Signal servers are compromised, and even if some

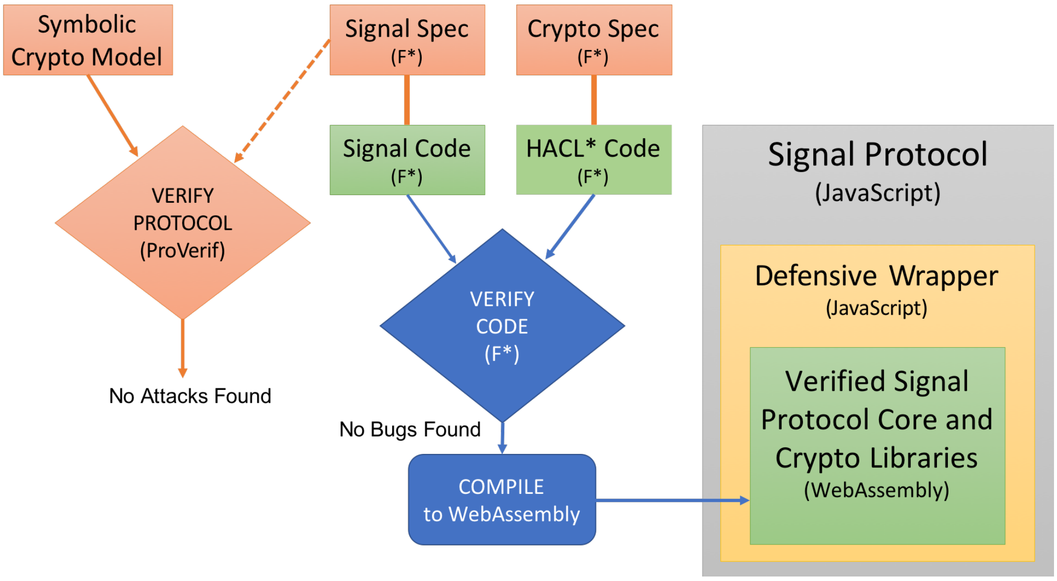


Figure 2 LibSignal* verification and compilation toolchain.

user devices are stolen or confiscated. To this end, Signal uses a novel key exchange protocol called X3DH [35] paired with an aggressive key update mechanism called Double Ratchet [38] that frequently changes message encryption keys, rendering old keys obsolete. Formally, Signal seeks to achieve a novel property called *post-compromise security* [21], in addition to classic secure channel guarantees like sender authentication, message confidentiality, and forward secrecy.

There are several implementations of Signal, including official libraries in Java (for Android phones), in C (for iPhones), and in JavaScript (for Web applications), that are embedded within various messaging applications. For example, the desktop version of Skype uses a library called libsignal-javascript for private conversations. This means that any flaw in the design of Signal or a bug in its JavaScript code may break the security of these private conversations.

We have built a verified implementation of Signal called LibSignal$*$ [40] using the tool chain depicted in Figure 2. Note the similarity in the overall work flow with our tool chain in Figure 1. We first write a formal specification of the Signal protocol in the pure fragment of F$*$. We then hand-translate this specification to the syntax of the ProVerif protocol analyzer [14] and verify it for all the target security properties of Signal, including forward and post-compromise security, following the methodology of [33]. If ProVerif fails to verify the protocol, it produces a counter-example that may indicate a security vulnerability. However, our analysis found no flaws in Signal, except for a known replay vulnerability [33].

Our next step was to write a Low$*$ implementation of Signal, which needed several cryptographic algorithms, including AES-CBC, HMAC, Curve25519, Ed25519, and SHA-2, all of which we implement and verify in HACL$*$. We then verify the Low$*$ code of Signal (composed with the Low$*$ code for HACL$*$) for conformance to the high-level protocol specification. Finally, we compile the code, via the KreMLin compiler to C and WebAssembly, obtaining verified implementations of the Signal protocol in these languages.

WebAssembly [31] is a new meta-assembly language supported by all Web browsers and many application frameworks. It allows compact and efficient low-level programs to be embedded within JavaScript applications and run on any platform. In comparison to

JavaScript, WebAssembly enjoys many advantages, making it a good target for verified code. In particular, WebAssembly is a small, statically typed language with a clean formal semantics, and it offers strong isolation guarantees against malicious JavaScript code. We develop a formal translation from Low∗ to WebAssembly and implement this as a new back-end for the KreMLin compiler [40]. We use this back-end to compile both HACL∗ and LibSignal∗. Our WebAssembly version of HACL∗ may independently be used in any JavaScript application that needs verified cryptography.

We observe that just generating the core cryptographic protocol code for Signal does not make it immediately usable by a messaging application. For example, the libsignal-protocol-javascript library provides a session and key management layer and exposes a simple interface to its applications. Our implementation of LibSignal∗ borrows this JavaScript code so that we meet the same interface and pass all the interoperability tests of Signal. Notably, however, we embed our verified WebAssembly code into the unverified JavaScript in a defensive manner that reduces the risk of private key exposure.

Our work in Signal is also influenced by our experience with the verification of the Transport Layer Security (TLS) protocol, the de facto standard for secure communications across the Internet. Although TLS was carefully specified and widely implemented, a large number of vulnerabilities were regularly found, both in the protocol design (e.g [13]) and in its implementations (e.g. [9]). When the Internet Engineering Task Force (IETF) began the process of standardizing TLS 1.3, it invited researchers to help them design the new protocol to be secure by design. Many researchers responded to this challenge, publishing a series of papers analyzing various draft versions of the protocol. In our work, we built detailed formal models of several drafts of TLS 1.3 using the verification tools ProVerif and CryptoVerif [10]. As part of Project Everest [11], we are also helping build a verified implementation of TLS 1.3 in F∗ using the same tool chain as HACL∗, but extending it with cryptographic security proofs [12].

Our work with LibSignal∗ and TLS shows how we can compose the low-level guarantees of HACL∗ with the sophisticated security proofs of ProVerif and other tools to obtain a verified cryptographic protocol implementation that can readily be deployed in real world messaging applications. We believe that this methodology offers a template for many more future applications.

4 Verified Applications

Encouraged by this flexibility and modularity, we plan to extend our framework to target distributed web applications beyond cryptographic primitives and communication protocols. We describe this next in the context of preserving privacy in machine learning classification, where its secure implementation will require certification of application code on clients and servers.

4.1 Privacy Preserving Machine Learning

Machine learning classification as-a-service is an attractive use-case for cloud servers. Such a server would host a classifier algorithm, and process and reply to classification queries from authenticated subscribers (or clients). Since learning applications consume large amounts of training data to generate useful classifiers, user privacy is a pressing concern. Protecting sensitive and personally identifiable information (PII) of users from servers, both during model learning and subsequent classification is desirable and can be a legal/compliance requirement. We focus only on the classification phase here to illustrate our techniques.

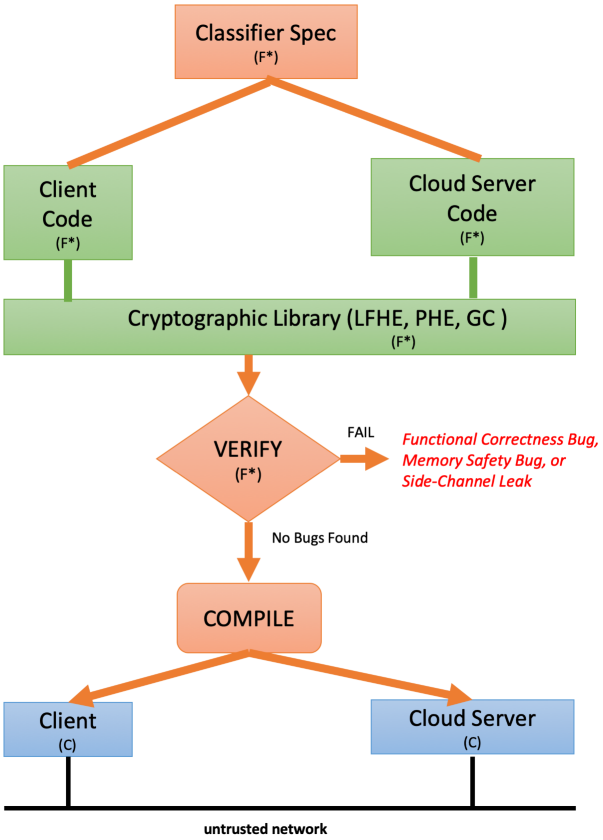


Figure 3 Programming and Verification Framework: The programmer first writes a high-level mathematical specification of the classifier (or any other computation over private data) in F∗. The programmer can run and test this specification. She then implements this specification as a distributed program with components running at the client and the cloud server. The program is composed with a cryptographic library and the whole system is verified using F∗. If verification succeeds, the code is compiled to C and can be deployed on the network.

A machine-learning classifier that preserves user privacy should not learn anything about the user query issued by a client or its resulting response (i.e.,the resulting class). At the same time, from the point of view of server, the mathematical models used for learning and inference can be proprietary and need to be hidden from clients.

We describe the context of our work in more detail. In model learning, the inputs to the learning algorithm are labeled data values, converted to feature vectors $\vec{x}$, and used to learn a model of weights w of a classifier consisting of say k classes $c_1 \cdots c_k$, given by $C(\vec{x}, w)$. In the *classification* phase, the label $c_j, 1 \leq j \leq k$ for an unseen feature vector $\vec{y}$ input by a client, is predicted using the classifier C as $c_j = C(\vec{y}, w)$. As mentioned earlier, we focus only on the classification phase, where the server is presented with a query and is expected to return the appropriate class label prediction.

Cryptographic techniques can offer a solution to this problem that satisfies both parties. Some relevant cryptographic schemes in this context include applications based on homomorphic encryption (HE) [27, 18, 32, 30] secure multi-party computation (SMC) [45, 34], garbled circuits [30, 32], and functional encryption (FE) [17], which allow clients and servers to jointly compute functions over encrypted or private data without revealing their inputs to each other. In HE, the result also remains encrypted, and can only be decrypted with the appropriate key. A typical HE algorithm takes an encrypted input x for program P and produces the encrypted result of applying x on the function encoded by P.

With HE, both the model w and query $\vec{y}$ are encrypted using say a public HE key. The prediction classifier is implemented on the server as the homomorphic evaluation function $Eval(C)$. The result of the prediction, c_j, has to be *declassified* and presented to the client that issued the query. The cryptographic properties of the HE scheme ensure that the client does not learn anything about model w beyond what it can learn from observing the predicted class of its input, and the server does not learn the value of the input, or its predicted class. A caveat here is that there are certain types of attacks, including model inversion, and access to prior knowledge, that can reveal user information even if they are encrypted. Techniques such as differential privacy [25] can help alleviate these concerns, and we plan to study them in the future.

HE schemes that can compute arbitrary functions (called fully HE or FHE) are fairly straightforward to implement, but are prohibitively expensive. Even with the latest implementation of HELib [28], general depth-limited homomorphic computations of interest in machine learning have very large overheads, e.g, with matrix multiplication being over 600K times slower than plaintext computations, which does not make them practical for useful applications. However, HE schemes that are restricted in their functionality, called partial HE schemes (PHE) are more practical, and can perform one type, say add or multiply [37, 29] or a small number of computations, e.g., quadratic functions [16]. We have seen e.g., in [18, 32], that PHE schemes can be combined with other auxiliary cryptographic schemes such as secure multi-party computations (SMC) and garbled circuits (GC), or even with strong hardware protection guarantees to build solutions that are practical, and provide strong guarantees.

We propose a programming and verification framework to help developers build distributed software applications using composite PHE protocols, and extend it to include auxiiary schemes such as SMC and GCs, incorporating verified cryptographic primitives and their high-performance implementations. With our framework, a developer can prove that the application code is functionally correct, that it correctly composes the various cryptographic schemes and protocols it uses, and that it does not accidentally leak any secrets (via side-channels, for example.) Our end-to-end solution can be seen as a logical extension of our work presented in the earlier two sections, and results in verified and efficient implementations of state-of-the-art secure privacy-preserving learning and classification.

Given a high-level algorithmic specification of a machine learning classifier, along with a set of confidentiality constraints on its inputs, our goal is to build and verify its implementation as an efficient distributed cryptographic application. Our verified implementation tool chain is shown in Figure 3, with four stages. Again, note how this can be seen as a modular extension of our earlier designs.

1. **Global High-Level Specification**: We first write a global high-level specification of our desired distributed computation in F∗, focusing on classification algorithms for now. The specification consists of the function ϕ it computes, the characterization of its model w, in terms of feature vectors $\vec{\chi}$, input $\vec{x} \in \vec{\chi}$, and the result $c_i = \phi(w, \vec{x})$ from $\mathcal{C}$ the set of classes. The high-level confidentiality specification is that the evaluation of ϕ must preserve the secrecy of w, $\vec{x}$, and c_i from relevant parties.
2. **Distributed Implementation**: We then write implementations, also in F∗, of the client and the cloud server, detailing all their network interactions and cryptographic computations. We prove that this implementation meets the high-level spec, while

preserving our desired confidentiality goals, given an abstract (trusted) interface for the underlying cryptography. The implementation can itself be broken into a reusable verified library of commonly used constructions, like addition, secure comparison, dot products, polynomial evaluation, etc. and application-specific code for the classification algorithm we seek to implement.

3. **Cryptographic Instantiation**: The code for these two parties will usually rely on a variety of cryptographic primitives, which will need to be instantiated with concrete schemes such as Paillier, GCs, random permutations, etc. which are themselves hard to implement correctly. We build verified implementations of all the cryptographic schemes we need, as an extension to the HACL∗ verified crypto library. These primitives compile to C code that is as fast as state-of-the-art hand-written crypto libraries. Each primitive is verified for memory safety, resistance to common timing side-channels, and functional correctness with respect to a high-level mathematical specification. We propose to build a series of verified HE and SMC schemes in HACL∗, which will also be reusable in other applications.
4. **Low-Level Executable Components**: Finally, we compile all our F∗ code along with the cryptographic library to C to obtain two C implementations, one for the client and one for the server. We envisage that these libraries will be embedded into larger applications that will handle less security-critical concerns like user interfaces, networking code, and persistent storage. Generating C code allows our code to run efficiently on a variety of platforms, including smartphones, and enables existing legacy applications to use our toolchain to verify their core cryptographic components.

At the end of this workflow, we obtain high performance verified protocol code in C for clients, and servers which can communicate over an untrusted network, but still provide strong correctness and confidentiality guarantees.

5 Proposed Roadmap

We propose to build our verification toolchain in stages, evaluating them over a series of case studies. Our eventual goal is to be able to verify privacy-preserving implementations of inference for naïve Bayes classifiers, hyperplane decision classifiers (perceptron, least squares, Fischer's linear discriminants, SVMs), decision tree classifiers, and neural networks.

As a longer-term goal, we see our toolchain as something that can be integrated into a mainstream framework for building distributed cryptographic applications. For example, the machine learning framework can be integrated with TensorFlow [5], which offers an API to developers that is not very far from the core operations we consider in this work: addition, multiplication, dot-product, comparison etc. We envision that machine learning developers will be able to write and test their high-level specifications as TensorFlow programs and our toolchain will help them develop verified low-level distributed protocols that implement these programs in a privacy-preserving style that can be safely deployed in the untrusted cloud.

Our verified framework and the modular tool chain allows us to develop high-assurance cryptographic applications that incorporate state-of-the-art cryptographic algorithms, complicated cryptographic protocols and their composition, and allow us analyze the resulting implementation for sophisticated and fine-grained end-to-end security properties.

References

1 Signal. `https://signal.org/docs/`.

2 ChaCha20 and Poly1305 for IETF Protocols. IETF RFC 7539, 2015.

3 Elliptic Curves for Security. IETF RFC 7748, 2016.

4 Edwards-Curve Digital Signature Algorithm (EdDSA) . IETF RFC 8032, 2017.

5 Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: `http://tensorflow.org/`.

6 José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Arthur Blot, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Hugo Pacheco, Benedikt Schmidt, and Pierre-Yves Strub. Jasmin: High-Assurance and High-Speed Cryptography. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1807–1823, 2017.

7 Andrew W Appel. Verification of a cryptographic primitive: SHA-256. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 37(2):7, 2015.

8 Daniel J Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In *International Conference on Cryptology and Information Security in Latin America (LATINCRYPT)*, pages 159–176. Springer, 2012.

9 Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A Messy State of the Union: Taming the Composite State Machines of TLS. In *IEEE Symposium on Security and Privacy*, pages 535–552, 2015.

10 Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *IEEE Symposium on Security and Privacy*, pages 483–503, 2017.

11 Karthikeyan Bhargavan, Barry Bond, Antoine Delignat-Lavaud, Cédric Fournet, Chris Hawblitzel, Catalin Hritcu, Samin Ishtiaq, Markulf Kohlweiss, Rustan Leino, Jay Lorch, Kenji Maillard, Jianyang Pang, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Ashay Rane, Aseem Rastogi, Nikhil Swamy, Laure Thompson, Peng Wang, Santiago Zanella-Béguelin, and Jean-Karim Zinzindohoué. Everest: Towards a Verified, Drop-in Replacement of HTTPS. In *Summit on Advances in Programming Languages (SNAPL)*, 2017.

12 Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jianyang Pan, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, and Jean Karim Zinzindohoué. Implementing and Proving the TLS 1.3 Record Layer. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2017.

13 Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. In *IEEE Symposium on Security and Privacy*, pages 98–113, 2014.

14 Bruno Blanchet. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, October 2016.

15 Barry Bond, Chris Hawblitzel, Manos Kapritsos, K. Rustan M. Leino, Jacob R. Lorch, Bryan Parno, Ashay Rane, Srinath Setty, and Laure Thompson. Vale: Verifying High-Performance Cryptographic Assembly Code. In *USENIX Security Symposium*, 2017.

16 Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography Conference (TCC)*, pages 325–341, 2005.

17 Dan Boneh, Amit Sahai, and Brent Waters. Functional Encryption: Definitions and Challenges. In *Theory of Cryptography Conference (TCC)*, pages 253–273, 2011.

18 Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data. In *Network and Distributed System Security Symposium (NDSS)*, 2015.

19 Billy B Brumley, Manuel Barbosa, Dan Page, and Frederik Vercauteren. Practical realisation and elimination of an ECC-related software bug attack. In *Topics in Cryptology (CT-RSA)*, pages 171–186. Springer, 2012.

20 Yu-Fang Chen, Chang-Hong Hsu, Hsin-Hung Lin, Peter Schwabe, Ming-Hsien Tsai, Bow-Yaw Wang, Bo-Yin Yang, and Shang-Yi Yang. Verifying Curve25519 Software. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 299–309, 2014.

21 Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On Post-compromise Security. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 164–178, 2016.

22 Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and Secure Comparison for On-Line Auctions. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Information Security and Privacy*, 2007.

23 Ivan Damgard, Martin Geisler, and Mikkel Kroigard. A Correction to “Efficient and Secure Comparison for On-Line Auctions”. *Int. J. Appl. Cryptol.*, 1(4), August 2008.

24 Jason A. Donenfeld. WireGuard: Next Generation Kernel Network Tunnel. In *Network and Distributed System Security Symposium (NDSS)*, 2017.

25 Cynthia Dwork. Differential Privacy. In *International Conference on Automata, Languages and Programming (ICALP) - Volume Part II*, 2006.

26 A. Erbsen, J. Philipoom, J. Gross, R. Sloan, and A. Chlipala. Simple High-Level Code for Cryptographic Arithmetic - With Proofs, Without Compromises. In *IEEE Symposium on Security and Privacy*, 2019.

27 Craig Gentry. *A Fully Homomorphic Encryption Scheme.* PhD thesis, Stanford University, Stanford, USA, 2009.

28 Craig Gentry and Shai Halevi. Implementing Gentry’s Fully-homomorphic Encryption Scheme. In *Advances in Cryptology (EUROCRYPT)*, pages 129–148, 2011.

29 Shafi Goldwasser and Silvio Micali. Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information. In *ACM Symposium on Theory of Computing (STOC)*, 1982.

30 Trinabh Gupta, Henrique Fingler, Lorenzo Alvisi, and Michael Walfish. Pretzel: Email Encryption and Provider-supplied Functions Are Compatible. In *Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 169–182, 2017.

31 Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the Web Up to Speed with WebAssembly. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 185–200, 2017.

32 Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security Symposium*, pages 1651–1669, 2018.

33 Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroSP)*, pages 435–450, 2017.

34 Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. EPIC: Efficient Private Image Classification (or: Learning from the Masters). Topics in Cryptology (CT-RSA), 2019.

35 Moxie Marlinspike and Trevor Perrin. The X3DH Key Agreement Protocol, 2016. URL: `https://signal.org/docs/specifications/x3dh/`.

36 Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238, 1999.

37 Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology (EUROCRYPT)*, pages 223–238, 1999.

38 Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm, 2016. URL: `https://signal.org/docs/specifications/doubleratchet/`.

39 Andy Polyakov, Ming-Hsien Tsai, Bow-Yaw Wang, and Bo-Yin Yang. Verifying Arithmetic Assembly Programs in Cryptographic Primitives. In *Conference on Concurrency Theory (CONCUR)*, 2018.

40 Jonathan Protzenko, Benjamin Beurdouche, Denis Merigoux, and Karthikeyan Bhargavan. Formally Verified Cryptographic Web Applications in WebAssembly. In *IEEE Symposium on Security and Privacy*, pages 1256–1274, 2019.

41 Jonathan Protzenko, Jean-Karim Zinzindohoué, Aseem Rastogi, Tahina Ramananandro, Peng Wang, Santiago Zanella-Béguelin, Antoine Delignat-Lavaud, Catalin Hritcu, Karthikeyan Bhargavan, Cédric Fournet, and Nikhil Swamy. Verified Low-Level Programming Embedded in F*. *PACMPL*, 1(ICFP):17:1–17:29, September 2017.

42 E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, 208. IETF RFC 8446.

43 Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoue, and Santiago Zanella-Béguelin. Dependent Types and Multi-Monadic Effects in F*. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 256–270, 2016.

44 National Institute of Standards US Department of Commerce and Technology (NIST). Federal Information Processing Standards Publication 180-4: Secure hash standard (SHS), 2012.

45 Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: Efficient and Private Neural Network Training. In *Privacy Enhancing Technologies Symposium.* (PETS 2019), 2019.

46 Jean Karim Zinzindohoue, Evmorfia-Iro Bartzia, and Karthikeyan Bhargavan. A Verified Extensible Library of Elliptic Curves. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 296–309, 2016.

47 Jean-Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. HACL*: A verified modern cryptographic library. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1789–1806, 2017.

Sketching Graphs and Combinatorial Optimization

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel
robert.krauthgamer@weizmann.ac.il

Abstract

Graph-sketching algorithms summarize an input graph G in a manner that suffices to later answer (perhaps approximately) one or more optimization problems on G, like distances, cuts, and matchings. Two famous examples are the Gomory-Hu tree, which represents all the minimum *st*-cuts in a graph G using a tree on the same vertex set $V(G)$; and the cut-sparsifier of Benczúr and Karger, which is a sparse graph (often a reweighted subgraph) that approximates every cut in G within factor $1 \pm \varepsilon$. Another genre of these problems limits the queries to designated terminal vertices, denoted $T \subseteq V(G)$, and the sketch size depends on $|T|$ instead of $|V(G)|$.

The talk will survey this topic, particularly cut and flow problems such as the three examples above. Currently, most known sketches are based on a graphical representation, often called edge and vertex sparsification, which leaves room for potential improvements like smaller storage by using another representation, and faster running time to answer a query. These algorithms employ a host of techniques, ranging from combinatorial methods, like graph partitioning and edge or vertex sampling, to standard tools in data-stream algorithms and in sparse recovery. There are also several lower bounds known, either combinatorial (for the graphical representation) or based on communication complexity and information theory.

Many of the recent efforts focus on characterizing the tradeoff between accuracy and sketch size, yet many intriguing and very accessible problems are still open, and I will describe them in the talk.

2012 ACM Subject Classification Theory of computation → Sketching and sampling; Mathematics of computing → Combinatorial optimization; Theory of computation → Graph algorithms analysis

Keywords and phrases Sketching, edge sparsification, vertex sparsification, Gomory-Hu tree, mimicking networks, graph sampling, succinct data structures

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.2

Category Invited Talk

Funding *Robert Krauthgamer*: Work partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, and a Minerva Foundation grant.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 2; pp. 2:1–2:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Finkel Was Right: Counter-Examples to Several Conjectures on Variants of Vector Addition Systems

Ranko Lazić
DIMAP, Department of Computer Science, University of Warwick, UK
R.S.Lazic@warwick.ac.uk

Abstract

Studying one-dimensional grammar vector addition systems has long been advocated by Alain Finkel. In this presentation, we shall see how research on those systems has led to the recent breakthrough tower lower bound for the reachability problem on vector addition systems, obtained by Czerwiński et al. In fact, we shall look at how appropriate modifications of an underlying technical construction can lead to counter-examples to several conjectures on one-dimensional grammar vector addition systems, fixed-dimensional vector addition systems, and fixed-dimensional flat vector addition systems.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Verification by model checking; Theory of computation → Program reasoning

Keywords and phrases Petri nets, vector addition systems, reachability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.3

Category Invited Talk

Funding *Ranko Lazić*: Partly supported by the Leverhulme Trust Research Fellowship "Petri Net Reachability Conjecture" (RF-2017-579).

Acknowledgements This presentation is based on joint work with Wojciech Czerwiński, Sławomir Lasota, Jérôme Leroux and Filip Mazowiecki. I also thank Matthias Englert for inspiring conversations about these variants of vector addition systems.

1 Outline

We shall discuss counter-examples to each of the following conjectures.

In the context of the gap between the ExpSpace membership of the coverability problem for one-dimensional grammar vector addition systems [2] and its PSpace hardness [5], it was attractive to think that there is an exponential bound such that, at least for systems whose ratio is finite and greater than 1, every positive instance has some derivation whose height is at most the bound.

▶ **Conjecture 1.** *One-dimensional grammar vector addition systems whose ratio is finite and greater than* 1 *have coverability witnessing derivations that are at most exponentially high.*

It has been known since the refinement by Rosier and Yen [4] of Rackoff's bounds [3] that fixed-dimensional vector addition systems have coverability witnessing runs that are at most exponentially long, and there seemed to be no reason why the same should not hold for the reachability problem.

▶ **Conjecture 2.** *Fixed-dimensional vector addition systems have reachability witnessing runs that are at most exponentially long.*

It has also been known that flat vector addition systems in which update vectors are given in unary have reachability witnessing runs that are at most polynomially long, provided the dimension is 1 or 2 [1], and it seemed plausible that this property would be true in every fixed dimension.

▶ **Conjecture 3.** *Fixed-dimensional flat vector addition systems in which update vectors are given in unary have reachability witnessing runs that are at most polynomially long.*

The same work of Englert et al. established NL membership of the reachability problem for two-dimensional flat unary vector addition systems, and it was similarly plausible that the same would be the case in every fixed dimension. However, we shall discuss how to obtain NP hardness already in a specific small dimension.

References

1 Matthias Englert, Ranko Lazić, and Patrick Totzke. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *LICS*, pages 477–484. ACM, 2016. doi:10.1145/2933575.2933577.

2 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the Coverability Problem for Pushdown Vector Addition Systems in One Dimension. In *ICALP, Part II*, volume 9135 of *LNCS*, pages 324–336. Springer, 2015. doi:10.1007/978-3-662-47666-6_26.

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

4 Louis E. Rosier and Hsu-Chun Yen. A Multiparameter Analysis of the Boundedness Problem for Vector Addition Systems. *J. Comput. Syst. Sci.*, 32(1):105–135, 1986. doi:10.1016/0022-0000(86)90006-1.

5 Juliusz Straszyński. Complexity of the reachability problem for pushdown Petri nets. Master's thesis, University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics, 2017. URL: https://apd.uw.edu.pl/diplomas/155747.

Progress in Lifting and Applications in Lower Bounds

Toniann Pitassi
University of Toronto, Canada
Institute for Advanced Study, Princeton, NJ, USA
toni@cs.toronto.edu

Abstract

Ever since Yao introduced the communication complexity model in 1979, it has played a pivotal role in our understanding of limitations for a wide variety of problems in Computer Science. In this talk, I will present the lifting method, whereby communication lower bounds are obtained by lifting much simpler lower bounds. I will show how lifting theorems have been used to solve many open problems in a variety of areas of computer science, including: circuit complexity, proof complexity, combinatorial optimization (size of linear programming formulations), cryptography (linear secret sharing schemes), game theory and privacy.

At the end of the talk, I will sketch the proof of a unified lifting theorem for deterministic and randomized communication (joint with Arkadev Chattopadyhay, Yuval Filmus, Sajin Koroth, and Or Meir.)

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases complexity theory, lower bounds, communication complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.4

Category Invited Talk

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 4; pp. 4:1–4:1
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

How Computer Science Informs Modern Auction Design

Tim Roughgarden
Columbia University, Department of Computer Science, New York, USA
tr@cs.columbia.edu

Abstract

Over the last twenty years, computer science has relied on concepts borrowed from game theory and economics to reason about applications ranging from internet routing to real-time auctions for online advertising. More recently, ideas have increasingly flowed in the opposite direction, with concepts and techniques from computer science beginning to influence economic theory and practice.

In this lecture, I will illustrate this point with a detailed case study of the 2016-2017 Federal Communications Commission incentive auction for repurposing wireless spectrum. Computer science techniques, ranging from algorithms for NP-hard problems to nondeterministic communication complexity, have played a critical role both in the design of the reverse auction (with the government procuring existing licenses from television broadcasters) and in the analysis of the forward auction (when the procured licenses sell to the highest bidder).

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases Game Theory, Auction Design, Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.5

Category Invited Talk

Funding Supported by NSF Award CCF-1813188 and ARO grant W911NF1910294.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 5; pp. 5:1–5:1
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An Algebraic Framework to Reason About Concurrency

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

Abstract

Kleene algebra with tests (KAT) is an algebraic framework for reasoning about the control flow of sequential programs. Hoare, Struth, and collaborators proposed a concurrent extension of Kleene Algebra (CKA) as a first step towards developing algebraic reasoning for concurrent programs. Completing their research program and extending KAT to encompass concurrent behaviour has however proven to be more challenging than initially expected. The core problem appears because when generalising KAT to reason about concurrent programs, axioms native to KAT in conjunction with expected axioms for reasoning about concurrency lead to an unexpected equation about programs. In this talk, we will revise the literature on CKA(T) and explain the challenges and solutions in the development of an algebraic framework for concurrency.

The talk is based on a series of papers joint with Tobias Kappé, Paul Brunet, Bas Luttik, Jurriaan Rot, Jana Wagemaker, and Fabio Zanasi [1, 2, 3]. Additional references can be found on the CoNeCo project website: `https://coneco-project.org/`.

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

Keywords and phrases Kleene Algebra, Concurrency, Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.6

Category Invited Talk

Supplement Material Additional references can be found on the CoNeCo project website: `https://coneco-project.org/`.

Funding *Alexandra Silva*: Research supported by ERC starting grant Profoundnet (679127).

References

1 Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva, and Fabio Zanasi. Brzozowski Goes Concurrent - A Kleene Theorem for Pomset Languages. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CONCUR.2017.25`.

2 Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Kleene Algebra with Observations. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory (CONCUR 2019)*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CONCUR.2019.41`.

3 Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. Concurrent Kleene Algebra: Free Model and Completeness. In Amal Ahmed, editor, *Programming Languages and Systems*, pages 856–882, Cham, 2018. Springer International Publishing.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 6; pp. 6:1–6:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Connected Search for a Lazy Robber

Isolde Adler
School of Computing, University of Leeds, Leeds, United Kingdom
I.M.Adler@leeds.ac.uk

Christophe Paul
CNRS, LIRMM, Université de Montpellier, Montpellier, France
christophe.paul@lirmm.fr

Dimitrios M. Thilikos
CNRS, LIRMM, Université de Montpellier, Montpellier, France
sedthilk@thilikos.info

Abstract

The node search game against a lazy (or, respectively, agile) invisible robber has been introduced as a search-game analogue of the treewidth parameter (and, respectively, pathwidth). In the *connected* variants of the above two games, we additionally demand that, at each moment of the search, the *clean* territories are *connected*. The connected search game against an agile and invisible robber has been extensively examined. The monotone variant (where we also demand that the clean territories are progressively increasing) of this game, corresponds to the graph parameter of *connected pathwidth*. It is known that *the price of connectivty* to search for an agile robber is bounded by 2, that is the connected pathwidth of a graph is at most twice (plus some constant) its pathwidth. In this paper, we investigate the connected search game against a *lazy* robber. A lazy robber moves only when the searchers' strategy threatens the location that he currently occupies. We introduce two alternative graph-theoretic formulations of this game, one in terms of connected tree decompositions, and one in terms of (connected) layouts, leading to the graph parameter of *connected treewidth*. We observe that connected treewidth parameter is closed under contractions and prove that for every $k \geq 2$, the set of contraction obstructions of the class of graphs with connected treewidth at most k is infinite. Our main result is a complete characterization of the obstruction set for $k = 2$. One may observe that, so far, only a few complete obstruction sets are explicitly known for contraction closed graph classes. We finally show that, in contrast to the agile robber game, the price of connectivity is unbounded.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Graph searching, Tree-decomposition, Obstructions

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.7

Funding *Christophe Paul*: Supported by projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).
Dimitrios M. Thilikos: Supported by projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010). Supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs, via the Franco-Norwegian project PHC AURORA 2019.

1 Introduction

A *graph-search game* is opposing a group of searchers and a robber that are moving in turn on a graph. A *search strategy* is a sequence of moves of the searchers that eventually leads to the capture of the robber. The *cost* of a search strategy is the maximum number of searchers simultaneously present on the graph during the search strategy. The *search number* of a graph is defined as the minimum cost of a search strategy. Different rules imposed on the search strategy and the moves of the robber define different searching games. The

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 7; pp. 7:1–7:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

study of graph searching parameters is an active field of graph theory as several important graph parameters have their search-game analogues that provide useful insights. For related surveys, see [2, 3, 10, 21, 38].

One of the most classic graph-search games is the one of *node-search* introduced by Kirousis and Papadimitriou [31, 32]. In this version, both the searchers and the robber occupy vertices of the graph. One searcher can move at a time. The capture of the robber happens when some searcher and the robber simultaneously occupy the same vertex and that the robber cannot escape along a path free of a searcher. In this paper we consider *monotone* search strategies against an *invisible* robber. Being invisible implies that the search strategy has to be independent of the moves of the robber. A search strategy is monotone if it prevents the robber from moving to vertices that have been already occupied by the searchers, implying that the *robber territory* is never increasing. The robber territory is the set of vertices that can be reached from the robber position by a path free of searcher.

Agility and laziness. A robber can be *lazy* or *agile*. A lazy robber resides on a vertex as long as a searcher is not placed on that vertex, while an agile robber may move whenever he wants to. The distinction between a lazy and an agile robber was introduced for the first time in [13]. Motivated by established links with well-studied graph theoretical parameters, there is an extensive amount of research on the different variants of the search game depending on the monotonicity constraint and on the laziness or agility of the robber. In particular, the monotone search number of a graph G against an agile (resp. lazy) robber is equal to the pathwidth (resp. treewidth) of G [13, 31, 32, 36, 42]. Also, it was proven that the non-monotone variants are equal to their monotone counterparts [8, 9, 20, 34, 42].

The connectivity issue. In both search games described above, no constraint (apart from the monotonicity, which in this context, as mentioned before, is no restriction) is ruling the move of a searcher. That is, a searcher can move arbitrarily far away from his/her original position. For this reason, such search games have been called "helicopter search games" (as suggested in [42]). From the application view point, this teleportation ability is not always realistic. In some settings (like cave exploration), it is natural to constrain the search to be connected. That is, the clean territory induces a connected subgraph[1] at each step of the search (see [24] for an example).

This inspired the question on the "price of connectivity", asking whether there is some universal constant c such that the connected search number is no more than c times its non-connected counterpart. In its original form, this question was asked in [5] for the agile variant and, in the same paper, it was answered affirmatively for the case of trees (see also [6,16–19,22,37] for related results). Later, it was proved for all graphs by Dereniowski [14], who suggested a connected counterpart of pathwidth, called connected pathwidth, that is equivalent to the monotone connected agile search number. Then it was proved that this parameter is always upper bounded by twice the pathwidth plus one.

[1] Interestingly, the motivating story of one of the foundational articles on graph searching, authored by Torrence Parsons [39] in 1976, was inspired by an earlier article of Breisch in *Southwestern Cavers Journal* [11] proposing a "speleotopological" approach for the problem of finding an explorer lost in a system of dark caves. It is worth to stress that this setting neglected the natural connectivity requirement.

1.1 Our contributions

Connected treewidth. In this paper, we study the (monotone) connected search against a *lazy* robber. Our first contribution is to establish the parameter by giving two alternative definitions: one in terms of *connected* tree decompositions and one in terms of *connected* layouts. Intuitively, a tree-decomposition $(T, \mathcal{F})$ is connected[2] if it can be rooted at some node r in a way that for every node u, the subgraph G_u, induced by the subset V_u of vertices appearing in some bag on the path in T between r and u, is connected. We observe that this is a natural extension of the concept of connected pathwidth proposed by [14]. Our layout definition is a variant of the classic layout definition of [13] with the restriction that now we only consider layouts where every prefix induces a connected graph. Our equivalence, proven in Section 3, indicates that monotone connected search against a *lazy* robber can be seen as a natural way to define a *connected* version of treewidth. We also stress that the non-monotone variant of this game corresponds to an different parameter, as proved in [24]. Yet another way to define "connected" treewidth is to consider tree decompositions where for every $t \in V_T$, the bag X_t induces a connected subgraph of G. We refer to this variant *bag-connected treewidth* (while the one we define in this paper can be called *prefix-connected treewidth*). Bag-connected treewidth was introduced independently by Jégou and Terrioux in [27], in the context of solving Constraint Satisfaction Problems (CSPs)(see [26, 28]) and, in a combinatorial context, by Diestel and Müller in [15] who revealed interesting relations with graph-geometric parameters such as the geodesic cycle number, graph hyperbolicity (see also [25]).

Contraction Obstructions. We say that a graph H is a *contraction* of G, denoted by $H \preceq G$, if a graph isomorphic to H can be obtained from G by a series of *edge contractions*. We also say that H is a *minor* of G if H is a contraction of a subgraph of G. We define the *minor obstructions* (*contraction obstructions*, respectively) of a graph class $\mathcal{G}$, denoted by $\mathbf{obs}_{\leq}(\mathcal{G})$ ($\mathbf{obs}_{\preceq}(\mathcal{G})$, respectively), as the set of all minor (contraction, respectively) minimal graphs that do not belong to $\mathcal{G}$. It is easy to see that when $\mathcal{G}$ is minor (contraction, respectively) closed, then $\mathbf{obs}_{\leq}(\mathcal{G})$ ($\mathbf{obs}_{\preceq}(\mathcal{G})$, respectively) provides a *complete* characterization of a minor closed (contraction, respectivelly) class $\mathcal{G}$: a graph belongs to $\mathcal{G}$ if and only if it excludes all graphs in $\mathbf{obs}_{\leq}(\mathcal{G})$ (respectively $\mathbf{obs}_{\preceq}(\mathcal{G})$) as minors (contractions, respectively). Moreover, in the case of the minor relation, we know from the theorem of Roberston and Seymour [40] that the set $\mathbf{obs}_{\leq}(\mathcal{G})$ is always finite and therefore the aforementioned characterization provides a *finite characterization* of any minor closed class in terms or forbidden minors. To identify (or even to compute) $\mathbf{obs}_{\leq}(\mathcal{G})$ for different instantiations of minor closed graph classes is an interesting topic in graph theory (see [1, 35]). For instance, if $\mathcal{T}_k$ is the class of graphs with treewidth at most k, then $\mathbf{obs}_{\leq}(\mathcal{T}_k)$ is known for every $k \leq 3$ [4] and remains unknown for $k > 3$ (see [41] for some partial results for the case where $k = 4$). Similarly, if $\mathcal{P}_k$ is the class of graphs with pathwidth at most k, then $\mathbf{obs}_{\leq}(\mathcal{P}_k)$ is known for $k \leq 2$ [30] and remains unknown for $k > 2$. Bounds for the size of the graphs in $\mathbf{obs}_{\leq}(\mathcal{T}_k)$ and $\mathbf{obs}_{\leq}(\mathcal{P}_k)$ have been proved in [33].

Unfortunately, the landscape is more obscure for the contraction relation as contraction obstruction sets are not finite in general. Contraction obstruction sets are only known for a few contraction closed classes. For instance, the contraction obstruction set for planar

[2] We also want to point out that alternative notions of connected tree-decomposition have been considered, see for example [23] and [15, 27] for two different definitions. We believe that the parameter correspondence we establish is a strong argument in favour of our definition proposal.

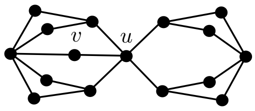


Figure 1 A graph $G \in \mathcal{T}_2^c$ such $G - uv \notin \mathcal{T}_2^c$ and $G - v \notin \mathcal{T}_2^c$.

graphs is described in [12]. A more elaborate example of a finite contraction obstruction set was identified in [7], containing 177 connected graphs, for the class of graphs whose connected mixed search number (for an agile and invisible robber) is at most 2. Another class characterized by an infinite set of contraction obstructions is discussed in [29].

Let $k \in \mathbb{N}$. By $\mathcal{T}_k^c$, we denote the class of all (connected) graphs with connected treewidth at most k. We observe that $\mathcal{T}_2^c$ is not minor closed: removing a vertex or an edge (see e.g., the graph G of Figure 1) may increase the connected treewidth. Therefore, no characterization via minor obstruction exists. However, in this paper we observe that $\mathcal{T}_k^c$ is contraction closed, for every k, and it is a challenging problem to identify $\mathcal{O}_k := \mathbf{obs}_{\preceq}(\mathcal{T}_k^c)$ for distinct values of k, especially since we have no guarantee that this set is finite. Moreover, in case $\mathcal{O}_k$ is infinite, we are essentially looking for a finite canonical description of this set.

Our second contribution is the complete identification of $\mathcal{O}_2$. As a preliminary part of our results, in Subsection 4.1, we prove general properties of $\mathcal{O}_k$ for every k. These are later used to identify $\mathcal{O}_2$. In Section 5, that is the most technical part of this paper, we prove that $\mathcal{O}_2$ is an infinite set that can be canonically described by a sequence of gluing operations.

Price of connectivity. We give, for every $k \geq 2$, an *infinite* subset of $\mathbf{obs}_{\preceq}(\mathcal{T}_k^c)$ consisting of graphs of treewidth 2, i.e., graphs in $\mathcal{T}_2$ (see Section 4.2). Consequently, the price of connectivity on treewidth is unbounded and this makes a sharp contrast with the corresponding result on pathwidth. To conclude, for monotone search, the price of connectivity is bounded when we are searching for an agile robber while this price goes to infinity when the robber is lazy. This latter contribution provides a simpler construction of a result from [24] that the cost of connectivity can be log n, where n is the number of vertices.

2 Preliminaries

2.1 Standard definitions

Sequences. Given a finite set U, a *sequence* σ over U is a bijection $\sigma : U \to [|U|]$. For $x \in U$, $\sigma(x) = i$ if x is at the i-th position in σ and we denote $\sigma_i = \sigma^{-1}(i)$. For $x, y \in U$, if $\sigma(x) < \sigma(y)$, we write $x <_\sigma y$. We define the sets $\sigma_{<i} = \{x \in U \mid \sigma(x) < i\}$ and $\sigma_{\leq i} = \{x \in U \mid \sigma(x) \leq i\}$. Alternatively, we denote a sequence by $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$.

Graphs. The graphs we consider are undirected are simple. We use standard notations. For a subset S of vertices, $G[S]$ denotes the subgraph induced by S. A *separator* is a subset S of vertices such that $G \setminus S = G[V \setminus S]$ contains more connected components than G. A connected component H of $G \setminus S$ is a *full S-component* of G if $N_G(V(H)) = S$. We denote by $\mathcal{C}(G, S)$ the set of all full S-connected components of G and by $\mathcal{F}(G, S)$ the set containing every induced subgraph $G[S \cup C]$ with $C \in \mathcal{C}(G, S)$. The set of cut vertices of a graph G is denoted $C(G)$.

Contracting an edge $e = xy \in E(G)$ yields the graph G/e obtained by removing x and y from G, introducing a new vertex and making it adjacent with all vertices in $N_G(\{x,y\})\setminus\{x,y\}$. If F is a subset of edges of G, then G/F is the graph obtained by contracting the edges of F. We say that a graph H is a *contraction* of G, denoted by $H \preceq G$, if a graph isomorphic to H can be obtained by a series of edge contractions.

A *tree-decomposition* of a graph $G = (V,E)$ is a pair $(T,\mathcal{F})$ where $T = (V_T, E_T)$ is a tree and $\mathcal{F} = \{X_t \subseteq V \mid t \in V_T\}$ such that : 1) $\bigcup_{t\in V_T} X_t = V$; 2) for every edge $e \in E$, there exists a node $t \in T$ such that $e \subseteq X_t$; and 3) for every vertex $x \in V$, the set $\{t \in V_T \mid x \in X_t\}$ induces a connected subgraph of T. We refer to V_T as the set of *nodes of* T and the sets of $\mathcal{F}$ as the *bags* of $(T,\mathcal{F})$. The *width* of a tree-decomposition $(T,\mathcal{F})$ is $\mathsf{width}(T,\mathcal{F}) = \max\big\{|X|-1 \mid X \in \mathcal{F}\big\}$ and the *tree-width* of a graph G is $\mathsf{tw}(G) = \min\big\{\mathsf{width}(T,\mathcal{F}) \mid (T,\mathcal{F})$ is a tree-decomposition of $G\big\}$.

Rooted graphs. A *q-rooted* graph (with $q \in \mathbb{N}$) is a pair $\mathbf{G} = (G,\mathbb{R})$ where G is a graph and $\mathbb{R}$ is a sequence over a subset R of q vertices of G, called *roots*. A *rooted graph* is any q-rooted graph, where $q \geq 0$. We treat every graph G as the 0-rooted graph $(G,\langle\rangle)$. The rooted graph $(G,\mathbb{R})$ is *connected* if either G is connected or if every connected component of G contains at least one vertex from $\mathbb{R}$. It is *biconnected* if adding an edge between every pair of root vertices yields a biconnected graph. *Gluing* two q-rooted graphs $(G_1,\mathbb{R}_1)$ and $(G_2,\mathbb{R}_2)$ results in the graph $(G_1,\mathbb{R}_1)\oplus(G_2,\mathbb{R}_2)$ obtained by identifying the vertex $\mathbb{R}_1(i)$ with $\mathbb{R}_2(i)$ for every $i \in [q]$. The operation of gluing $k \geq$ copies of a rooted graph $\mathbf{K}$ is denoted by $k \times \mathbf{K}$ and is defined in the obvious way (keep always in mind that the result is a graph). A rooted graph $\mathbf{H} = (H,\mathbb{T})$ is a contraction of a rooted graph $\mathbf{G} = (G,\mathbb{R})$, denoted $\mathbf{H} \preceq \mathbf{G}$ if a rooted graph isomorphic to $(H,\mathbb{T})$ can be obtained after a series of edge contractions on G, under the constraint that no path between two vertices of $\mathbb{R}$ can be contracted to a single vertex. If a vertex $v \in V(H)$ results from the contraction of an edge incident to a root vertex of $\mathbb{R}$, then v is a root vertex of $\mathbb{T}$.

Tree vertex separation. A *layout* σ of a rooted graph $\mathbf{G} = (G,\mathbb{R})$ is a sequence over $V(G)$ such that for every $1 \leq j \leq |\mathbb{R}|$, $\sigma^{-1}(j) \in \mathbb{R}$. We denote by $\mathcal{L}(\mathbf{G})$ the set of all layouts of $\mathbf{G}$. For every $i \in [n]$, the *supporting set* of position i is the set $S_\sigma(i) = \big\{x \in V(G) \mid \sigma(x) < i$ and there exists a (x,σ_i)-path whose internal vertices belong to $\sigma_{>i}\big\}$. The so-called *tree vertex separation number* of a rooted graph $\mathbf{G}$ is defined as $\mathsf{tvs}(\mathbf{G}) = \min\big\{\mathsf{tcost}(\mathbf{G},\sigma) \mid \sigma \in \mathcal{L}(G)\big\}$, where $\mathsf{tcost}(\mathbf{G},\sigma) = \max\{\big|S_\sigma(i)\big| \mid i \in [n]\}$.

Search strategies against a lazy robber. A *search strategy on* a graph G is a sequence $\mathcal{S} = \langle S_1,\ldots,S_r\rangle$, with $r \in \mathbb{N}$, over the sets of subsets of vertices of $V(G)$ where $|S_1| = 1$ and for all $i \in [r-1]$, the symmetric difference of S_i and S_{i+1} has cardinality one. Notice that the difference between two consecutive set either corresponds to a placement or to the removal of a searcher on some vertex v. The *cost* of a search strategy $\mathcal{S}$ is $\mathsf{cost}(\mathcal{S}) = \max\{|S_i| \mid i \in [r]\}$.

For a search strategy $\mathcal{S}$ against a lazy robber, we define the *sequence of robber spaces* as the sequence $\mathcal{F}_\mathcal{S} = \langle F_1,\ldots,F_r\rangle$ where:

- $F_1 = V(G) \setminus S_1$.
- For $i \in [2,r]$, let $F_i = (F_{i-1} - S_i) \cup \{v \in V - S_i$: there is a path from a vertex $u \in F_{i-1} \cap (S_i - S_{i-1})$ to v whose vertices except u belong to $V - S_i\}$.

The complementary sequence $\overline{\mathcal{F}}_\mathcal{S} = \langle \overline{F}_1,\ldots,\overline{F}_r\rangle$ is the sequence of *clean* spaces. We say that the search strategy $\mathcal{S}$ is *complete*, if $F_r = \emptyset$; *monotone*, if for each $i \in [r-1]$, $F_{i+1} \subseteq F_i$. We define $\mathsf{lns}(G)$ as the minimum cost of a complete (or, alternativaly, *cop-win*) monotone search strategy on G against a lazy robber.

3 Parameter equivalences

▶ **Proposition 1** ([13, 42]). *For any graph G, we have* $\mathsf{tw}(G) = \mathsf{tvs}(G) = \mathsf{lns}(G) - 1$.

To prove a result similar to the above well-known theorem, we adapt the definitions of graph search, tree decomposition, layouts and the associated parameters to the connected setting[3].

- A (monotone and complete) search strategy $\mathcal{S} = \langle S_1, \ldots, S_r \rangle$ of a graph G is *connected* if at every step $i \in [r]$ the clean space $\overline{F}_i$ is connected. We define the parameter $\mathsf{mclns}(G)$ as the minimum cost of a monotone, complete and connected strategy on G against a lazy robber.
- A tree-decomposition $(T, \mathcal{F})$ of a graph G is *connected* if there exists a node $r \in V(T)$ such that for every node $u \in V(T)$, the subgraph $G[V_u]$ is connected, where V_u contains all the vertices that belong to some bag X_t associated with a node t in the u, r-path in T. We then define the *connected treewidth* $\mathsf{ctw}(G)$ as the minimum width of a connected tree-decomposition. Figure 2 provides an example where the treewidth and the connected treewidth of a graph differs.
- A layout σ of a graph G is *connected* if for every $i \in [n]$, the subgraph $G[\sigma_{\leq i}]$ is connected. We let $\mathcal{L}^c(G)$ denote the set of connected layouts of G. We then define the *connected tree vertex separation* parameter as $\mathsf{ctvs}(G) = \min\{\mathsf{tcost}(G, \sigma) \mid \sigma \in \mathcal{L}^c(G)\}$.

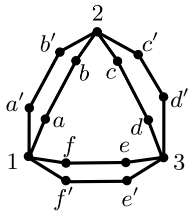


Figure 2 A series-parallel graph G with $\mathsf{tw}(G) = 2$ and $\mathsf{ctw}(G) = 3$. A connected tree-decomposition of minimum width is given by the path-decomposition $(P, \mathcal{F})$ where $V(P) = \{x_1, \ldots x_8\}$ and $\mathcal{F} = \{X_1 = \{1, a, b, 2\}, X_2 = \{1, a', b', 2\}, X_3 = \{1, 2, c, d\}, X_4 = \{1, 2, d, 3\}, X_5 = \{1, 2, 3, c'\}, X_6 = \{1, 3, c', d'\}, X_7 = \{1, 3, e, f\}, X_8 = \{1, 3, e', f'\}\}$, the root node being x_1.

Let us now state the main theorem of this section.

▶ **Theorem 2.** *For every connected graph G, we have $ctw(G) = ctvs(G) = mclns(G) - 1$.*

We stress that if in the proof above we use connected path decompositions instead of connected tree decompositions, we obtain the counterpart of Theorem 2 linking the connected path-width of a graph to the connected search number against an agile robber and to a parameter called *connected path vertex separation number.*

4 General properties of obstructions

A graph class $\mathcal{G}$ is closed under contraction, if every graph H, that is a contraction of a member G of $\mathcal{G}$, also belongs to $\mathcal{G}$. Assume $\mathcal{G}$ is closed under contraction, then a graph G is a *contraction obstruction* to $\mathcal{G}$, if $G \notin \mathcal{G}$ but $H \in \mathcal{G}$ for every $H \preceq G$. Similarly, a graph parameter $\kappa(\cdot)$ is closed under contraction if for every pair of graphs H and G such that $H \preceq G$, $\kappa(H) \leq \kappa(G)$.

[3] These definitions naturally extend to rooted graphs.

The following lemma, stated in terms of rooted graphs, proves that the parameters $\mathsf{ctw}(\cdot)$, $\mathsf{mclns}(\cdot)$ and $\mathsf{ctvs}(\cdot)$ are closed under contraction.

▶ **Lemma 3.** *Let $(G_1, \mathbb{R}_1)$ and $(G_2, \mathbb{R}_2)$ be two q-rooted graphs such that $(G_1, \mathbb{R}_1) \preceq (G_2, \mathbb{R}_2)$. Then $\mathsf{ctvs}(G_1, \mathbb{R}_1) \leq \mathsf{ctvs}(G_2, \mathbb{R}_2)$.*

4.1 Non-biconnected obstructions

We extend the notion of obstruction sets to rooted graphs in the natural manner. For every $q \geq 1$, we let $\mathcal{O}_k^{(q)}$ denoted the set containing every q-rooted graph $\mathbf{G} = (G, \mathbb{R})$, where $\mathsf{ctvs}(\mathbf{G}) > k$ and for every proper contraction G' of G, $\mathsf{ctvs}(G', \mathbb{R}') \leq k$. We can prove that an obstruction contains at most one cut vertex, meaning that knowing the set of biconnected obstructions and of 1-rooted obstructions will be enough to describe the full set of obstructions.

We now introduce some concepts on graphs. A vertex subset $S \subseteq V(G)$ is a *separator* if $G \setminus S = G[V \setminus S]$ contains more connected components than G. A connected component H of $G \setminus S$ is a *full S-component* of G if $N_G(V(H)) = S$. We denote by $\mathcal{C}(G, S)$ the set of all full S-connected components of G. We denote by $\mathcal{F}(G, S)$ the set containing every induced subgraph $G[S \cup C]$ with $C \in \mathcal{C}(G, S)$. A separator S is a *minimal separator* if $|\mathcal{F}(G, S)| \geq 2$. A minimal separator S is a *minimal $\langle x, y\rangle$-separator* if x and y belong to different full S-components. A vertex $x \in V(G)$ is a *cut-vertex* if $\{x\}$ is a separator. The set of cut vertices of a graph G is denoted $C(G)$. A graph G is *biconnected* if it is connected and $C(G) = \emptyset$. A *biconnected component* of a graph is any biconnected subgraph of G that is vertex-maximal. Let $x \in C(G)$ be a cut vertex of G. The pair (G, x) is called a *s-pair*. If $Z \in \mathcal{F}(G, \{x\})$, then the 1-rooted graph $(Z, \langle x\rangle)$ is a *1-component* of the s-pair (G, x). Similarly, if $\{x, y\}$ is a minimal separator of G, then the triple (G, x, y) is called a *s-triple*. A 2-rooted graph $(H, \langle x, y\rangle)$ is a *2-component* of the s-triple (G, x, y) if $\{x, y\}$ is a minimal separator of G and $H \in \mathcal{F}(G, \{x, y\})$.

A vertex v of a graph G is called *k-simplicial* if it has degree at most k and its neighborhood induces a complete subgraph. The proof of the next lemma is presented in Section 4 of the (attached) full version.

▶ **Lemma 4.** *For every $k \geq 1$ and every connected graph G, $G \in \mathcal{O}_k$ is not biconnected iff G contains exactly one cut vertex and $G \in \{\boldsymbol{A} \oplus \boldsymbol{B} \mid \boldsymbol{A}, \boldsymbol{B} \in \mathcal{O}_k^{(1)}\}$.*

The proof of Lemma 4 is a consequence of Lemma 3 and the next two Lemmas .

▶ **Lemma 5.** *If a connected graph G contains a k-simplicial vertex v, then $G \notin \mathcal{O}_k$.*

Proof. (sketch) The argument simply follows from the observation that $G' = G - v$ is a contraction of G and that extending a connected layout σ' of G' by adding v as the last vertex yields a connected layout σ of G such that $\mathsf{tcost}(G, \sigma) = \mathsf{tcost}(G', \sigma')$. ◀

▶ **Lemma 6.** *Let G be a connected graph. If $G \in \mathcal{O}_k$ and contains a cut vertex v, then the s-pair (G, v) contains exactly two 1-components and v is the unique cut vertex of G.*

Proof. (sketch) Suppose that v is a cut vertex of $G \in \mathcal{O}_k$ and that C_0, C_1, C_2 are distinct connected components of the graph $G - v$. It follows from Lemma 3 that for every $i \in \{0, 1, 2\}, \mathsf{ctvs}(G[C_i \cup C_{(i+1) \text{mod } 3} \cup \{v\}]) \leqslant k$, which implies that for every $i \in \{0, 1, 2\}$, $\mathsf{ctvs}(G[C_i \cup \{v\}], \{v\}) \leq k$ or $\mathsf{ctvs}(G[C_{(i+1) \bmod 3} \cup \{v\}], \{v\}) \leq k$. Using the connected layouts that certifies these later inequalities, one can build a connected layout σ of G such that $\mathsf{tcost}(G, \sigma) \leq k$, a contradiction to the fact that $G \in \mathcal{O}_k$.

So removing a cut vertex in G leaves exactly two connected components. Suppose that there exist two cut vertices x and y and let C_x (resp. C_y) be the connected component of $G - x$ not containing y (resp. of $G - y$ not containing x). Then applying arguments similar as the ones above to the subgraphs $G_x = [C_x \cup \{x\}]$, $G_y = G[C_y \cup \{y\}]$ and $G_{xy} = G - (C_x \cup C_y)$ allows to show the existence of a connected layout σ of G such that $\mathsf{tcost}(G, \sigma) \leq k$, a contradiction to the fact that $G \in \mathcal{O}_k$. ◀

4.2 On the price of connectivity

We next examine the question of the price of connectivity for connected treewidth. Let us recall that it is known that the connected pathwidth is a most twice the pathwidth of a graph. Concerning treewidth, as a consequence of Proposition 1 and of the proof of Proposition 7 below, we know that there exists graphs of treewidth at most 4 with abritrary large connected treewidth. Moreover increasing the connected treewidth by one requires to double the number of vertices.

▶ **Proposition 7** ([24]). *For any n_0, there is $n \geq n_0$ and an n-vertex graph G such that $\mathit{mclns}(G) \in \Omega(\mathit{mlns}(G) \cdot \log n)$.*

We strengthen the theorem above by proving that this result also holds when restricting to series-parallel graphs (that are biconnected graphs of treewidth at most two). Our construction yields to way more simpler graphs than in [24]. The proof of the next result is in Section 6 of the full version.

▶ **Theorem 8** (Corollary 2 in the full version). *For every $k \in \mathbb{N}$, the obstruction set $\mathbf{obs}(\mathcal{T}^c_k)$ contains infinitely many series-parallel graphs.*

To prove Theorem 8, we construct an infinite family of series-parallel graphs with arbitrarily large connected treewidth. For $k \geq 2$, we define the family $\mathcal{Q}_k = \{\mathbf{A} \oplus \mathbf{B} \mid \mathbf{A}, \mathbf{B} \in \mathcal{Y}_k\}$ where $\mathcal{Y}_k$ is the family of 1-rooted graphs $\mathbf{Y}_k = (Y_k, \langle r\rangle)$ that can be constructed as follows: take any tree T_k, rooted at vertex r, such that the distance between every leaf and r is k and every non-leaf vertex has at least two children; add an *apex* vertex z universal to the leaves of T_k; if r has only two neighbors, these neighbors may or may not be adjacent to each another.

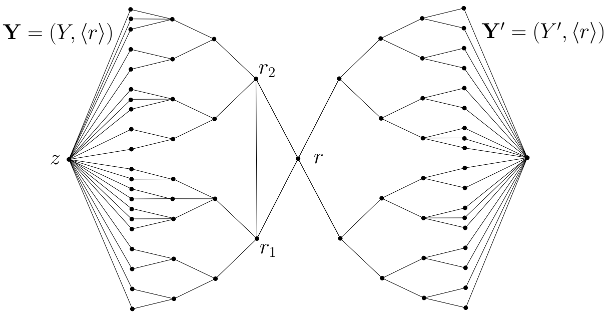

$\mathbf{Y} = (Y, \langle r\rangle)$
$\mathbf{Y}' = (Y', \langle r\rangle)$
r_2
z
r
r_1

Figure 3 A graph $H = \mathbf{Y} \oplus \mathbf{Y'} \in \mathcal{O}_4$ with $\mathbf{Y} = (Y, \langle r\rangle) \in \mathcal{Y}_4$ and $\mathbf{Y'} = (Y', \langle r\rangle) \in \mathcal{Y}_4$.

Theorem Theorem 8 is based on the following lemma.

▶ **Lemma 9.** *For every $k \geq 2$, $\mathcal{Q}_k \subseteq \boldsymbol{obs}_{\preceq}(\mathcal{T}^c_k)$.*

Proof. (sketch) We first observe that every 1-rooted graph $\mathbf{Y} = (Y, \langle r \rangle) \in \mathcal{Y}_k$ can be constructed from two (or more) graphs $\mathbf{Y}_1 = (Y_1, \langle r_1 \rangle) \in \mathcal{Y}_{k-1}$ and $\mathbf{Y}_2 = (Y_2, \langle r_2 \rangle) \in \mathcal{Y}_{k-1}$ by identifying their apex vertices and adding a root vertex r adjacent to the roots r_1 and r_2 of $\mathbf{Y}_1$ and $\mathbf{Y}_2$ (see Figure 3). For any $\mathbf{Y}_k = (Y_k, \langle r \rangle) \in \mathcal{Y}_k$, we define the 2-rooted graphs $\mathbf{Y}_k^{(2)} = (Y, \langle r, z \rangle)$ where z is the apex vertex of $\mathbf{Y}_k$. The proof relies on the following claims, that for every $k \geq 2$: (a) $\mathsf{ctvs}(\mathbf{Y}_k^{(2)}) = k$, (b) $\mathsf{ctvs}(\mathbf{Y}_k) > k$, and (c) for every edge e of Y_k, $\mathsf{ctvs}((Y_k/e, \langle r \rangle)) \leq k$.

Let us sketch the argument of the second claim. Consider a connected layout $\sigma \in \mathcal{L}^c(\mathbf{Y}_k)$ and suppose that $\sigma_i = z$. By the connectivity of σ, the induced subgraph $Y_k[\sigma_{\leq i}]$ contains a path P from the root r to the apex z. Observe that P contains exactly $k+2$ vertices $r, v_2, \dots v_{k+1}, z$ and that Y_k contains $k+1$ internally vertex disjoint paths from the apex z to $r, v_2, \dots v_{k+1}$. It follows that the supporting set $S_\sigma(i)$ contains at least $k+1$ vertices.

From the second and the third claims we conclude that every $\mathbf{Y}_k = (Y_k, \langle r \rangle) \in \mathcal{Y}_k$ belongs to the set $\mathcal{O}_k^1$ of 1-rooted obstructions of $\mathcal{T}_k^c$. By Lemma 4, we conclude that $\mathcal{Q}_k \subseteq \mathbf{obs}_{\preceq}(\mathcal{T}_k^c)$. ◀

5 The obstruction set $\mathcal{O}_2$

Thanks to Lemma 4, the non-biconnected parts of $\mathcal{O}_2$ can be determined if we identify the 1-rooted obstruction set $\mathcal{O}_2^{(1)}$. To that aim, let us first define the family $\mathcal{B}_2^{(1)} = \{\mathbf{Y}_x\} \cup \{\mathbf{Y}_x^{(k)} \mid k \geq 2]\}$ where $\mathbf{Y}_x^{(k)} = (k \times \mathbf{R}_x^y, \langle x \rangle)$ (see Figure 4). It is not difficult to check that these graphs are 1-rooted obstructions.

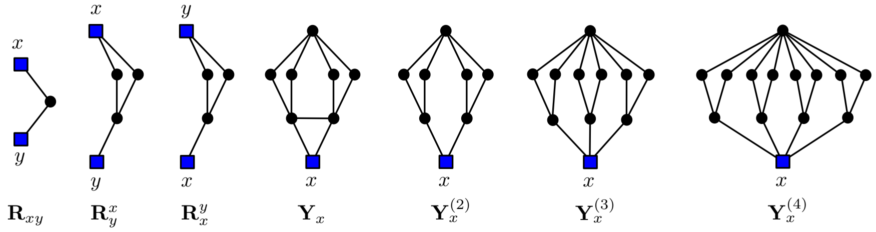


Figure 4 The rooted graphs $\mathbf{R}_{xy}$, $\mathbf{R}_x^y$, $\mathbf{R}_y^x$, $\mathbf{Y}_x$, $\mathbf{Y}_x^{(2)}$, $\mathbf{Y}_x^{(3)}$, and $\mathbf{Y}_x^{(4)}$.

It can be easily checked that the three biconnected graphs depicted in Figure 5 belong to $\mathcal{O}_2$. We define the set $\mathcal{B}_2 = \{K_4, W_1, W_2\} \cup \{\mathbf{A} \oplus \mathbf{B} \mid \mathbf{A}, \mathbf{B} \in \mathcal{B}_2^{(1)}\}$.

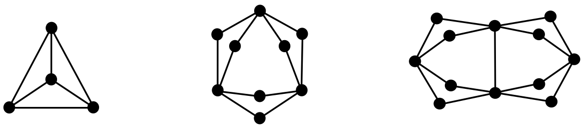


Figure 5 From left to right, the graphs K_4, W_1, and W_2.

2-twin expansion. Let $\mathbf{G} = (G, \mathbb{R})$ be a rooted graph and let $S \subseteq V(G)$. We say that S is a 2-*twin family* of $\mathbf{G}$ if $S \cap V(\mathbb{R}) = \emptyset$, $|S| \geq 2$ and there are two vertices $a, b \in V(G)$ such that $\forall s \in S, N_G(s) = \{a, b\}$. We call the vertices a, b the *bases* of the 2-twin family S. We say that a graph $\mathbf{G'} = (G', \mathbb{R}')$ is a 2-*twin expansion of* G if $\mathbb{R} = \mathbb{R}'$ and G' is obtained from G by adding vertices such that each additional vertex is made adjacent with the base vertices of some of the 2-twin families of $\mathbf{G}$. Given a class of rooted graphs $\mathcal{C}$ we define its 2-*twin expansion* $\mathsf{texp}(\mathcal{C})$ as the class of rooted graphs containing all 2-twin expansions of all the graphs in $\mathcal{C}$. We are now ready to state the main result of this section.

▶ **Theorem 10.** *A graph G belongs to $\mathcal{T}_2^c$ if and only if it does not contains a graph of $\mathsf{texp}(\mathcal{B}_2)$ as a contraction, that is $\mathcal{O}_2 = \mathsf{texp}(\mathcal{B}_2)$.*

5.1 Some elements of the proof of Theorem 10

The set $\mathcal{O}_2$ is closed under twin expansion. We say that a rooted graph $\mathbf{G}$ is *simplified* if all its 2-twin families have size 2. Given a rooted graph $\mathbf{G}$ we denote by $\tilde{\mathbf{G}}$ the unique simplified rooted graph such that $\mathbf{G} \in \mathsf{texp}(\{\tilde{\mathbf{G}}\})$. Given a set $\mathcal{C}$ of rooted graphs, we define $\tilde{\mathcal{C}} = \{\tilde{\mathbf{G}} \mid \mathbf{G} \in \mathcal{C}\}$. Observe that every graph of $\mathcal{B}_2$ is simplified.

▶ **Lemma 11.** *A graph G belongs to $\mathcal{O}_2$ iff $\tilde{G}$ belongs to $\mathcal{O}_2$.*

The next lemma is the extension of Lemma 11 to 1-rooted obstructions. Its proof mainly follows from Lemma 4.

▶ **Lemma 12.** *Let $\mathbf{H} = (H, \langle v \rangle)$ be a 1-rooted graph. Then $\mathbf{H} \in \mathcal{O}_2^{(1)}$ if and only if $\tilde{\mathbf{H}} \in \tilde{\mathcal{O}}_2^{(1)}$.*

Simplified obstructions. We now identify sets of simplified graphs, 1-rooted graphs and 2-rooted graphs that are obstructions. Later, we prove that from these sets the full set of obstructions to $\mathcal{T}_2^c$ can be constructed. The following Lemmas establish that the sets $\mathcal{B}_2^{(1)}$ and $\mathcal{B}_2$ build from the graphs of Figure 4 and Figure 5 are simplified obstructions.

▶ **Lemma 13.** *If a 1-rooted graph $\boldsymbol{G} = (G, \langle x \rangle)$ belongs to $\mathsf{texp}(\mathcal{B}_2^{(1)})$, then $\mathbf{G} \in \mathcal{O}_2^{(1)}$. If a graph G belongs to $\mathsf{texp}(\mathcal{B}_2)$, then G belongs to $\mathcal{O}_2$.*

Let us now turn to biconnected 2-rooted obstructions. We define the set $\mathcal{B}_2^{(2)} = \{\mathbf{R}^{xy}, \mathbf{R}^{xy+}, \mathbf{K}_4^{xy-}, \mathbf{K}_4^{xy}\}$ of 2-rooted graphs depicted in Figure 6. We say that a biconnected 2-rooted graph $\mathbf{H} = (H, \langle x, y \rangle)$ is *elementary* if it is $\mathsf{texp}(\mathcal{B}_2^{(2)})$-free.

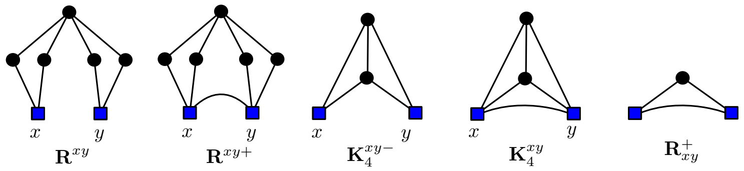


Figure 6 The 2-rooted graphs $\mathbf{R}^{xy}, \mathbf{R}^{xy+}, \mathbf{K}_4^{xy-}, \mathbf{K}_4^{xy}$ and $\mathbf{R}^+_{xy}$.

▶ **Lemma 14.** *The set of biconnected graphs in $\mathcal{O}_2^{(2)}$ is $\mathsf{texp}(\mathcal{B}_2^{(2)})$.*

Proof. (sketch) By considering a minimal counter-example, the proof first establishes that every elementary biconnected 2-rooted graph belongs to $\mathcal{T}_2^c$. Then we check that for every 2-rooted graph $\mathbf{G} \in \mathsf{texp}(\mathcal{B}_2^{(2)})$, $\mathsf{ctw}(\mathbf{G}) \geq 3$ but every contraction of $\mathbf{G}$ belongs to $\mathcal{T}_2^c$. ◀

Structure of obstructions. Let xy be an edge of a graph G. We say that xy is a *separating edge* if the set $\{x, y\}$ is a minimal separator. We say that xy is a *marginal edge* if there is a vertex z such that both (G, x, z) and (G, y, z) are s-triples.

▶ **Lemma 15.** *Let G be a graph in $\tilde{\mathcal{O}}_2$. If G contains a separating edge xy, then either G is isomorphic to W_2 or G contains a cut-vertex r and the 1-component of the s-pair (G, r) containing xy is isomorphic to $\mathbf{Y}_r$.*

▶ **Lemma 16.** *Let $G \in \tilde{\mathcal{O}}_2$ be a graph without separating edge. If (G, x, y) is a s-triple, then*

1. *either every 2-component of (G, x, y) is elementary,*
2. *or there exists a non-elementary 2-component of (G, x, y), denoted by $\boldsymbol{H} = (H, \langle x, y\rangle)$, such that $G \setminus (V(H) \setminus \{x, y\})$ cannot be contracted to $\ell \times \boldsymbol{R}_{xy}$ for any $\ell \geq 2$.*

From Lemma 15 and Lemma 16, we deduce a series of properties needed to understand the role of marginal edges and to conclude the characterization of $\mathcal{O}_2$.

▶ **Lemma 17.** *Let (G, x, y) be an s-triple of $G \in \tilde{\mathcal{O}}_2$. If $(H, \langle x, y\rangle)$ is a 2-component of (G, x, y) that is isomorphic to $\boldsymbol{R}^y_x$, then x is a cut-vertex.*

▶ **Lemma 18.** *Let (G, x, y) be an s-triple of $G \in \tilde{\mathcal{O}}_2$. If $\boldsymbol{H}$ is an elementary 2-component of (G, x, y) without cut-vertex, then $\boldsymbol{H}$ is isomorphic to $\boldsymbol{R}_{xy}$.*

▶ **Lemma 19.** *Let $\boldsymbol{G} = (G, \langle x\rangle) \in \tilde{\mathcal{O}}_2^{(1)}$.*

1. *If (G, x, y) is a s-triple, then none of its 2-components is isomorphic to $\boldsymbol{R}_{xy}$.*
2. *If $\boldsymbol{H} = (H, \langle x, y\rangle)$ is an elementary 2-component of an s-triple (G, x, y), then $\boldsymbol{H}$ is isomorphic to $\boldsymbol{R}^y_x$.*

Biconnected obstructions. We now have all the ingredients for the proof of Theorem 10. We start with the identification of the biconnected elements of $\tilde{\mathcal{O}}_2$.

▶ **Lemma 20.** *No biconnected graph in $\tilde{\mathcal{O}}_2$ contains a marginal edge.*

▶ **Lemma 21.** *The biconnected graphs in $\tilde{\mathcal{O}}_2$ are the graphs K_4, W_1, and W_2.*

Proof. (sketch) For a contradiction, we suppose that $\tilde{\mathcal{O}}_2$ contains a graph G distinct from K_4, W_1, and W_2. From Lemma 15, G does not contain a separating edge. As it excludes K_4 as a contraction, it contains a degree-two vertex a. Let x and y be the two neighbors of a and let $\mathcal{H} = \{\mathbf{H}_0, \ldots, \mathbf{H}_q\}$ be the 2-components of the s-triple (G, x, y) with $V(H_0) = \{a, x, y\}$. As G is biconnected, so is every 2-rooted graph in $\mathcal{H}$. We next prove that exactly one of the 2-rooted graphs in $\{\mathbf{H}_1, \ldots, \mathbf{H}_q\}$, say $\mathbf{H}_1$, is not elementary. Then by Lemma 18, every $\mathbf{H}_j \in \mathcal{H}$ distinct from $\mathbf{H}_1$ is isomorphic to $\mathbf{R}_{xy}$. As G is simplified, we have $q \leq 2$. If $q = 2$, as $\mathbf{H}_0 \oplus \mathbf{H}_2 = 2 \times \mathbf{R}_{xy}$, as $\mathbf{H}_1$ is not elementary and as G does not contains a separating edge, Lemma 16 leads to a contradiction. In the case $q = 1$, it can be proved that H_1 contains a cut vertex, implying the existence of a marginal edge in G, a contradiction to Lemma 20. ◀

Non-biconnected obstructions. The second part of the proof of Theorem 10 identifies the non-biconnected elements of $\tilde{\mathcal{O}}_2$.

▶ **Lemma 22.** *The non-biconnected graphs in $\tilde{\mathcal{O}}_2$ are the graphs in $\{\boldsymbol{A} \oplus \boldsymbol{B} \mid \boldsymbol{A}, \boldsymbol{B} \in \mathcal{B}_2^{(1)}\}$.*

Proof. (sketch) From Lemma 4 and Lemma 13, it is enough to prove that $\tilde{\mathcal{O}}_2^{(1)} \subseteq \mathcal{B}_2^{(1)}$. We assume, towards a contradiction, that there is some 1-rooted graph $\mathbf{G} = (G, \langle r\rangle) \in \tilde{\mathcal{O}}_2^{(1)} \setminus \mathcal{B}_2^{(1)}$. Observe that $\mathbf{G}$ is $\mathcal{B}_2^{(1)}$-free and G is biconnected. From Lemma 15, we can assume that G does not have separating edges. Let $J = 2 \times \mathbf{G}$. As the underlying graphs of the 2-rooted graphs in $\mathcal{B}_2^{(1)}$ are $\{K_4, W_1, W_2\}$-free, Lemma 4 implies that J is $\mathcal{B}_2$-free and thereby K_4-free. It can easily be seen that r has more than two neighbors. Also one may consider a 2-tree T that contains G as a spanning subgraph and satisfies the following properties

- (D1) If an edge is marginal in T then it is also marginal in G.
- (D2) If an edge is simplicial in T then one of its endpoints have degree 2 in G.
- (D3) If an edge is a separating edge of G, then it is also a separating edge in T.

Let z be a neighbor of r. Because of (D3), the edge $e = rz$ is either a marginal or a simplicial edge of T. We claim that e is marginal. Indeed, if e is simplicial, then from (D2) z has degree 2. Let w be the other neighbor of z. Notice that one of the 2-components of the s-triple (G, r, w) is isomorphic to $\mathbf{R}_{rw}$, a contradition to Lemma 19. We now know that $e = rz$ is a marginal edge. Let t be the base of e. cleanly (G, r, t) is an s-triple and $tr \notin E(G)$ as G does not have separating edges. We denote by $\mathcal{U} = \mathbf{U}_1, \ldots, \mathbf{U}_q$ the 2-components of (G, r, t). Our next step is to prove that all 2-rooted graphs in $\mathcal{U}$ are simple. This, together with Lemma 19 imply that all graphs in $\mathcal{U}$ are isomorphic to $\mathbf{R}^t_r$. This means that $\mathbf{G}$ contains as a contraction some $\mathbf{Y}^{(\ell)}_t$ for some $\ell \geq 3$. As each such $\mathbf{Y}^{(\ell)}_t$ belongs to $\mathcal{B}^{(1)}_2$ we have a contradiction. ◀

References

1 Isolde Adler. Open Problems related to computing obstruction sets. Manuscript, September 2008.

2 Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous.* International Series in Operations Research & Management Science, 55. Kluwer Academic Publishers, Boston, MA, 2003.

3 Brian Alspach. Searching and sweeping graphs: a brief survey. *Matematiche (Catania)*, 59(1-2):5–37 (2006), 2004.

4 Stefan Arnborg, Andrzej Proskurowski, and Derek G. Corneil. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics*, 80(1):1–19, 1990.

5 Lali Barrière, Paola Flocchini, Fedor V. Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, and Dimitrios M. Thilikos. Connected graph searching. *Inf. Comput.*, 219:1–16, 2012. doi:10.1016/j.ic.2012.08.004.

6 Lali Barrière, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M. Thilikos. Searching Is Not Jumping. In *29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2003)*, volume 2880 of *LNCS*, pages 34–45. Springer, 2003.

7 Micah J Best, Arvind Gupta, Dimitrios M. Thilikos, and Dimitris Zoros. Contraction obstructions for connected graph searching. *Discrete Applied Mathematics*, 209:27–47, 2016. 9th International Colloquium on Graph Theory and Combinatorics, 2014, Grenoble.

8 D. Bienstock and Paul Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.

9 Dan Bienstock, Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a forest. *J. Comb. Theory Ser. B*, 52(2):274–283, 1991.

10 Daniel Bienstock. Graph searching, path-width, tree-width and related problems (a survey). In *Reliability of computer and communication networks (New Brunswick, NJ, 1989)*, volume 5 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 33–49. Amer. Math. Soc., Providence, RI, 1991.

11 R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers (A publication of the Southwestern Region of the National Speleological Society)*, VI(5):72–78, 1967.

12 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic Graph Minor Theory: Improved Grid Minor Bounds and Wagner's Contraction. *Algorithmica*, 54(2):142–180, 2009. doi:10.1007/s00453-007-9138-y.

13 Nick D. Dendris, Lefteris M. Kirousis, and Dimitrios M. Thilikos. Fugitive-search games on graphs and related parameters. *Theoret. Comput. Sci.*, 172(1-2):233–254, 1997.

14 Dariusz Dereniowski. From Pathwidth to Connected Pathwidth. *SIAM J. Discrete Math.*, 26(4):1709–1732, 2012. doi:10.1137/110826424.

15 Reinhard Diestel and Malte Müller. Connected Tree-Width. *Combinatorica*, 38(2):381–398, 2018. doi:10.1007/s00493-016-3516-5.

16 Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Contiguous Search in the Hypercube for Capturing an Intruder. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005) IPDPS*. IEEE Computer Society, 2005.

17 Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Decontamination of chordal rings and tori using mobile agents. *Int. J. of Found. of Comp. Sc.*, 18(3):547–564, 2007.

18 Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Decontamination of hypercubes by mobile agents. *Networks*, page to appear, 2007.

19 F. V. Fomin, D. M. Thilikos, and I. Todinca. Connected Graph Searching in Outerplanar Graphs. *Electronic Notes in Discrete Mathematics*, 22:213–216, 2005. 7th International Colloquium on Graph Theory. Short communication.

20 Fedor V. Fomin and Dimitrios M. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Appl. Math.*, 131(2):323–335, 2003. Submodularity. doi:10.1016/S0166-218X(02)00459-6.

21 Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoret. Comput. Sci.*, 399(3):236–245, 2008.

22 Pierre Fraigniaud and Nicolas Nisse. Connected Treewidth and Connected Graph Searching. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN 2006)*, volume 3887 of *LNCS*, pages 479–490. Springer, 2006.

23 Pierre Fraigniaud and Nicolas Nisse. Connected Treewidth and Connected Graph Searching. In *7th Latin American Symposium on Theoretical Informatics (LATIN 2006)*, volume 3887 of *LNCS*, pages 479–490. Springer, 2006.

24 Pierre Fraigniaud and Nicolas Nisse. Monotony Properties of Connected Visible Graph Searching. In *32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2006)*, volume 4271 of *LNCS*, pages 229–240. Springer, 2006.

25 Matthias Hamann and Daniel Weißauer. Bounding Connected Tree-Width. *SIAM J. Discrete Math.*, 30(3):1391–1400, 2016. doi:10.1137/15M1044618.

26 Philippe Jégou and Cyril Terrioux. Bag-Connected Tree-Width: A New Parameter for Graph Decomposition. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2014, Fort Lauderdale, FL, USA, January 6-8, 2014*, 2014. URL: http://www.cs.uic.edu/pub/Isaim2014/WebPreferences/ISAIM2014_Jegou_Terrioux_New.pdf.

27 Philippe Jégou and Cyril Terrioux. Tree-Decompositions with Connected Clusters for Solving Constraint Networks. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 407–423. Springer, 2014. doi:10.1007/978-3-319-10428-7_31.

28 Philippe Jégou and Cyril Terrioux. Combining restarts, nogoods and bag-connected decompositions for solving CSPs. *Constraints*, 22(2):191–229, 2017. doi:10.1007/s10601-016-9248-8.

29 Marcin Kaminski, Daniël Paulusma, and Dimitrios M. Thilikos. Contracting planar graphs to contractions of triangulations. *J. Discrete Algorithms*, 9(3):299–306, 2011. doi:10.1016/j.jda.2011.03.010.

30 Nancy G. Kinnersley and Michael A. Langston. Obstruction set Isolation for the Gate Matrix Layout Problem. *Discrete Applied Mathematics*, 54:169–213, 1994.

31 Lefteris M. Kirousis and Christos H. Papadimitriou. Interval graphs and searching. *Discrete Math.*, 55(2):181–184, 1985.

32 Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theoret. Comput. Sci.*, 47(2):205–218, 1986.

33 J. Lagergren. Upper bounds on the size of obstructions and intertwines. *J. Comb. Theory, Ser. B*, 73:7–40, 1998.

34 Andrea S. LaPaugh. Recontamination does not help to search a graph. *J. Assoc. Comput. Mach.*, 40(2):224–245, 1993.

35 Thomas W. Mattman. Forbidden Minors: Finding the Finite Few. In Aaron Wootton, Valerie Peterson, and Christopher Lee, editors, *A Primer for Undergraduate Research: From Groups and Tiles to Frames and Vaccines*, pages 85–97, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-66065-3_4.

36 Rolf H. Möhring. Graph problems related to gate matrix layout and PLA folding. In *Computational graph theory*, volume 7 of *Comput. Suppl.*, pages 17–51. Springer, Vienna, 1990.

37 Nicolas Nisse. Connected graph searching in chordal graphs. *Discrete Applied Mathematics*, 157(12):2603–2610, 2009. Second Workshop on Graph Classes, Optimization, and Width Parameters. doi:10.1016/j.dam.2008.08.007.

38 Nicolas Nisse. *Network Decontamination*, pages 516–548. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-11072-7_19.

39 T. D. Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, volume 642 of *Lecture Notes in Math.*, pages 426–441. Springer, Berlin, 1978.

40 Neil Robertson and P. D. Seymour. Graph Minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

41 Daniel P. Sanders. On linear recognition of tree-width at most four. *SIAM J. Discrete Math.*, 9(1):101–117, 1996.

42 P. D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory Ser. B*, 58(1):22–33, 1993.

Parameterized Streaming Algorithms for Min-Ones d-SAT

Akanksha Agrawal
Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, Israel
akanksha.agrawal.2029@gmail.com

Arindam Biswas
The Institute of Mathematical Sciences, HBNI,
Chennai, India
arindam.b@ftml.net

Édouard Bonnet
CNRS, ENS de Lyon,
Université Claude Bernard Lyon 1
LIP UMR5668, France
edouard.bonnet@ens-lyon.fr

Nick Brettell
Department of Computer Science,
Durham University, Durham, UK
nbrettell@gmail.com

Radu Curticapean
BARC, University of Copenhagen,
Copenhagen, Denmark
ITU Copenhagen, Copenhagen, Denmark
radu.curticapean@gmail.com

Dániel Marx
Institute for Computer Science and Control,
MTA SZTAKI, Budapest, Hungary
dmarx@cs.bme.hu

Tillmann Miltzow
Department of Computer Science,
Utrecht University, Utrecht, Netherlands
t.miltzow@googlemail.com

Venkatesh Raman
The Institute of Mathematical Sciences, HBNI,
Chennai, India
vraman@imsc.res.in

Saket Saurabh
The Institute of Mathematical Sciences, HBNI,
Chennai, India
Department of Computer Science,
University of Bergen, Bergen, Norway
saket@imsc.res.in

Abstract

In this work, we initiate the study of the Min-Ones d-SAT problem in the parameterized streaming model. An instance of the problem consists of a d-CNF formula F and an integer k, and the objective is to determine if F has a satisfying assignment which sets at most k variables to 1. In the parameterized streaming model, input is provided as a stream, just as in the usual streaming model. A key difference is that the bound on the read-write memory available to the algorithm is $O(f(k)\log n)$ ($f : \mathbb{N} \to \mathbb{N}$, a computable function) as opposed to the $O(\log n)$ bound of the usual streaming model. The other important difference is that the number of passes the algorithm makes over its input must be a (preferably small) function of k.

We design a $(k+1)$-pass parameterized streaming algorithm that solves Min-Ones d-SAT ($d \geq 2$) using space $O\big((kd^{ck} + k^d)\log n\big)$ ($c > 0$, a constant) and a $(d+1)^k$-pass algorithm that uses space $O(k \log n)$. We also design a streaming kernelization for Min-Ones 2-SAT that makes $(k+2)$ passes and uses space $O\big(k^6 \log n\big)$ to produce a kernel with $O\big(k^6\big)$ clauses.

To complement these positive results, we show that any k-pass algorithm for Min-Ones d-SAT ($d \geq 2$) requires space $\Omega\big(\max\big\{n^{1/k}/2^k, \log(n/k)\big\}\big)$ on instances (F, k). This is achieved via a reduction from the streaming problem POT Pointer Chasing (Guha and McGregor [ICALP 2008]), which might be of independent interest. Given this, our $(k+1)$-pass parameterized streaming algorithm is the best possible, inasmuch as the number of passes is concerned.

In contrast to the results of Fafianie and Kratsch [MFCS 2014] and Chitnis et al. [SODA 2015], who independently showed that there are 1-pass parameterized streaming algorithms for Vertex Cover (a restriction of Min-Ones 2-SAT), we show using lower bounds from Communication

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 8; pp. 8:1–8:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Complexity that for any $d \geq 1$, a 1-pass streaming algorithm for MIN-ONES d-SAT requires space $\Omega(n)$. This excludes the possibility of a 1-pass parameterized streaming algorithm for the problem. Additionally, we show that any p-pass algorithm for the problem requires space $\Omega(n/p)$.

2012 ACM Subject Classification Theory of computation → Streaming models; Theory of computation → Fixed parameter tractability; Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Combinatorial algorithms

Keywords and phrases min, ones, sat, d-sat, parameterized, kernelization, streaming, space, efficient, algorithm, parameter

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.8

1 Introduction

The satisfiability problem (SAT) is among most studied NP-complete problems and serves as the canonical problem for NP, being the first problem which was shown to be NP-complete [6, 23]. It is an important problem in both theory and practice, and together with its variants, it appears in nearly every domain of Computer Science (see for example [9, 15, 16, 17, 30]). Because of this, the problem has been studied in various paradigms such as classical Complexity Theory [3], Approximation Algorithms [22, 34], Exact Algorithms [14, 32], Parameterized Complexity [7, 31], and Heuristics [16].

A variant which frequently appears in the literature is d-SAT ($d \geq 1$), where problem instances have at most d variables per clause. While d-SAT is NP-complete for $d \geq 3$, 2-SAT is a classic example of a tractable, i.e. polynomial-time-solvable problem. In this work, we study an optimization version of d-SAT in the framework of parameterized streaming, which combines streaming algorithms and parameterized algorithms.

The streaming framework was formulated to study the behaviour of algorithms that process large amounts of data in a sequential manner. The input appears as a sequence of items and the assumption is that the amount of read-write memory available to the algorithm is very limited, typically logarithmic in the total size of the input. Because of this, the algorithm is unable to store the entirety of its input in memory, and since the input appears in a sequence, the algorithm does not have random access to the it. It may however make multiple passes over the input. The goal in the streaming framework is to process the input by making as few passes (ideally, just one) over it as possible while using as little memory as possible. The study of problems in this framework dates back to the 1980s [12, 28], although the framework was formally established only in 1996 [2, 20]. The other player in the combined framework that we employ is Parameterized Complexity – an approach pioneered by Downey and Fellows [8]. For details on Parameterized Complexity, we refer the reader to the books of Downey and Fellows [8], Flum and Grohe [13], Niedermeier [29], and the recent book of Cygan et al. [7]. Appendix A provides a short introduction to the subject.

Min-Ones d-Sat and the Parameterized Streaming Model. We study the following optimization version of d-SAT which, among other things, generalizes VERTEX COVER and d-HITTING SET. For $d \geq 1$, the problem is defined as follows.

MIN-ONES d-SAT **Parameter:** k

Instance: (F, k), where F is a boolean formula with at most d literals per clause, and $k \in \mathbb{N}$.

Question: Can F be satisfied by setting at most k of its variables to 1?

It should be noted here that the problem 2-SAT admits a polynomial-time algorithm [4, 10, 25]. Its minimization version however, being a generalization of Vertex Cover is NP-hard [33]. Indeed, the graph in a Vertex Cover instance can be seen as a formula in which the vertices are variables and each edge is a monotone clause containing the two endpoints as (positive) literals.

Fafianie and Kratsch [11] considered the question of kernelizing d-Hitting Set, d-Set Matching and Edge Dominating Set in the streaming model. Chitnis et al. [5] studied the problems Maximal Matching and Vertex Cover in the parameterized streaming model. The space used by these algorithms is $\mathrm{O}(f(k)\log n)$, where k is the parameter, n is the size of the input, and $f : \mathbb{N} \to \mathbb{N}$ is a computable function.

The parameterized streaming model relaxes the space constraint of the usual streaming model to $f(k)\log n$, and allows the algorithm to make at most $g(k)$ passes over its input, where $g : \mathbb{N} \to \mathbb{N}$ is a (preferably slowly-growing) computable function. The goal now is to make as few passes over the input as possible, relative to the parameter. Under these new constraints, it is possible to construct streaming algorithms that have more refined space requirements, and we can also perform a more delicate analysis of the streaming complexity of the problem in question. Our results here illustrate this fact.

Our Results. In Section 2, we describe a parameterized streaming algorithm for Min-Ones d-SAT ($d \geq 2$) that solves instances (F, k) using $\mathrm{O}\big((kd^{ck} + k^d)\log n\big)$ ($c > 0$, a constant) bits of space and makes $k+1$ passes. We then show that by carefully simulating the execution stack of the standard branching algorithm for Min-Ones d-SAT, a $(d+1)^k$-pass, $\mathrm{O}(k\log n)$-space algorithm can be obtained. We believe that such an approach will be useful in the design of parameterized streaming algorithms for other problems as well. As an application, we show how the two algorithms can be used to solve IP_2 (a restricted Integer Programming problem) in the parameterized streaming model.

Section 3 describes a streaming kernelization for Min-Ones 2-SAT and an application of the algorithm to IP_2. By making $k+2$ passes over the input formula, it produces a kernel with $\mathrm{O}\big(k^6\big)$ clauses while using $\mathrm{O}\big(k^6\log n\big)$ bits of space. It is known that for $d \geq 3$, Min-Ones d-SAT does not admit a polynomial kernel [24] under certain (fairly reasonable) assumptions, ruling out a generalization of this result to larger values of d. Our algorithm also provides an alternative to the known kernelization [27] for the problem, since it can also be executed in the less restrictive random-access machine (RAM) model.

We then exhibit various lower bounds in Section 4 to complement the positive results above. For $d \geq 2$, we show that any k-pass streaming algorithm for Min-Ones-d-SAT requires $\Omega\big(\max\big\{n^{1/k}/2^k, \log(n/k)\big\}\big)$ bits of space in the worst case. This result is obtained by combining a well-known lower bound for the DISJ_k [26] problem from Communication Complexity and a lower bound for the streaming problem POT Pointer Chasing [18]. This (unconditional) lower bound implies, among other things, that the $k+1$ pass Min-Ones d-SAT ($d \geq 2$) algorithm of section 2 is pass-optimal.

The next result in the section shows that even for $d = 1$, any 1-pass algorithm for Min-Ones d-SAT requires space $\Omega(n)$. This is in contrast to the results of Fafianie and Kratsch [11] and Chitnis et al. [5], who independently showed that there are 1-pass parameterized streaming algorithms for the Vertex Cover problem (a restriction of Min-Ones 2-SAT). Finally, we show that any p-pass algorithm for Min-Ones d-SAT ($d \geq 1$), where p may be a function of both n and k, requires space $\mathrm{O}(n/p)$.

▶ Note 1.1. Although we do not provide an explicit accounting of the time used by our algorithms, it is not difficult to see that the streaming FPT algorithms all run in FPT time overall and the kernelizations, in polynomial time.

Related Results. Min-Ones 2-SAT was first studied by Gusfield and Pitt [19], who gave a polynomial-time 2-approximation algorithm for the problem. Misra et al. [27] exhibited an equivalence between Min-Ones-2 AT and Vertex-Cover via a polynomial-time parameter-preserving reduction. Fafianie and Kratsch [11], and Chitnis et al. [5] showed that Vertex Cover admits a single-pass, $\mathrm{O}(k^2)$-space algorithm. As noted earlier, Min-Ones 2-SAT generalizes Vertex Cover. Analogously, Min-Ones d-SAT generalizes d-Hitting Set. The question of kernelizing d-Hitting-Set was studied by Abu-Khzam [1], and Fafianie and Kratsch [11], who gave a single-pass algorithm that produces a kernel with $\mathrm{O}(k^d)$ sets.

Preliminaries. Here we introduce some basic concepts and notation used in the rest of the paper. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. Let $x \in \{0,1\}^n$ and $i \in [n]$. The i^{th} coordinate of x is denoted by $x[i]$. Consider a set of variables $V = \{x_1, \ldots, x_n\}$. A *literal* is a variable x_i (called an *unnegated* literal) or its negation $\neg x_i$ (called a *negated* literal). A *clause* is a disjunction (OR) of literals, e.g. $(x_1 \vee \neg x_2 \vee \neg x_3)$. It is called *monotone* if it consists entirely of unnegated literals, and is called *anti-monotone* if it consists entirely of negated literals. Clauses containing both negated and unnegated literals are called *non-monotone.*

A conjunction (AND) of clauses is called a CNF *formula.* When each clause has at most d literals, it is called a d-CNF formula. An *assignment* for a CNF formula F over the variable set V is a subset $S \subseteq V$. The assignment satisfies a clause if there is a variable in S that appears unnegated in the clause or a variable in $V \setminus S$ that appears negated in the clause. An assignment which satisfies all clauses in a formula is called a *satisfying* assignment for the formula.

2 Streaming FPT Algorithms

The main result of this section is an algorithm that solves instances (F, k) of Min-Ones d-SAT in $k+1$ passes using space $\mathrm{O}\big((kd^{ck} + k^d)\log n\big)$ ($c > 0$, a constant). We also describe how to simulate the execution of the standard branching algorithm for the problem to solve in instances in $(d+1)^k$ passes using space $\mathrm{O}(k \log n)$ (see Appendix B.1). Using these algorithms as subroutines, we then show how IP_2, a restricted version of the Integer Programming problem, where every constraint has at most two variables, can be solved in the parameterized streaming model (see Appendix D).

The $(k+1)$-pass algorithm begins by a making a single pass over the formula and obtains a set of minimal assignments for certain "essential" monotone clauses in the formula. In the next $k-1$ passes, these assignments are extended as much as possible using the implications appearing in the formula. Finally, the algorithm makes an additional pass to check if the formula as a whole is satisfied by one of the extended assignments.

Let $(\mathcal{F}, k)$ be an instance of Min-Ones 2-SAT on the variable set $V = \{x_1, x_2, \ldots, x_n\}$. The next result shows how a streaming kernelization for d-Hitting Set (defined below) can be used to enumerate minimal solutions for a certain hitting set problem.

d-Hitting Set **Parameter:** k

Instance: $(U, \mathcal{F}, k)$, where $\mathcal{F}$ is a family of subsets of U of size at most d, and $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq U$ of size at most k such that $S \cap A \neq \emptyset$ for all $A \in \mathcal{F}$?

▶ **Proposition 2.1** (♠[1]). *There is an algorithm Enum-d-HS, that finds the set $\mathcal{S}_k$, of all minimal d-hitting sets of size at most k, for an instance $I = (\mathcal{X}, U, k)$ of d-Hitting Set in time $\mathrm{O}\big(d^k|I|\big)$. Moreover, $|\mathcal{S}_k| \in \mathrm{O}\big(d^k\big)$ and the algorithm uses space $\mathrm{O}\big(k|I| + kd^k b_U\big)$, where $|I|$ is the size of I and b_U is maximum size of the elements of U in bits.*

The following result follows from Observation 1, Theorem 1 and Lemma 7 of [11].

▶ **Proposition 2.2.** *There is a 1-pass streaming algorithm called Stream-HS for d-Hitting-Set, which given an instance $I = (\mathcal{X}, U, k)$ with u_{max} as the maximum element of U, returns an (equivalent instance) $I' = (\mathcal{X}', U' \subseteq U, k)$ using $\mathrm{O}\big(k^d \log |U|\big)$ bits of memory and $\mathrm{O}\big(k^d\big)$ time at each step, such that the following conditions are satisfied.*

1. *$|\mathcal{X}'| \in \mathrm{O}\big(k^d\big)$ and the bit size of I' is bounded by $\mathrm{O}\big(k^d \log |U|\big)$.*
2. *Elements of U' are represented using $\log |U|$ bits.*
3. *$S \subseteq U$ (or U') of size at most k is a solution to I if and only if it is a solution to I'.*

We note that in item 1 of Proposition 2.2, the size of I' can be bounded by $\mathrm{O}\big(k^d \log k\big)$, by relabeling, but we want to preserve the exact variables, so we do not use relabeling.

Next, we apply the algorithm Stream-HS of Proposition 2.2 to obtain a set, which we call a set of *essential monotone clauses*, $\mathcal{C}_1$, and the set $\mathcal{S}_1$ of all minimal assignments (as sets of variables set to 1) for them of size at most k, as follows.

Pass 1. For each monotone clause $C = (x_1 \vee x_2 \vee \cdots \vee x_{d'})$ (where $d' \leq d$) seen in the stream, pass the set $\{x_1, x_2, \ldots, x_{d'}\}$ to Stream-HS. Let $I_t = (\mathcal{X}_t, U_t, k)$ be the output of Stream-HS once the entire stream has been read. Set $\mathcal{C}_1 = \mathcal{X}_t$. Using Proposition 2.1, compute the set $\mathcal{S}_1$, of all minimal d-hitting sets of size at most k for I_t.

The next lemma bounds the time and the space used in Pass 1.

▶ **Lemma 2.3** (♠). *Pass 1 uses space $\mathrm{O}\big((k^d + d^k)k \log n\big)$ and time $\mathrm{O}\big(d^k k^d \log n\big)$ after reading each clause from the stream.*

Let $\mathcal{C}^+$ be the set of all monotone clauses of $\mathcal{F}$, let $\mathcal{F}^+ = \wedge_{C \in \mathcal{C}^+} C$ and $\mathcal{F}_1^+ = \wedge_{C \in \mathcal{C}_1} C$. Recall that $\mathcal{C}_1$ is the set of clauses computed in Pass 1. We have the following observation, which follows from Proposition 2.1 and item 3 of Proposition 2.2.

▶ **Observation 2.4.** $\mathcal{S}_1$ is the set of all minimal satisfying assignments of size at most k for both $\mathcal{F}^+$ and $\mathcal{F}_1^+$.

The next observation relates satisfying assignments to $\mathcal{F}$ and the family $\mathcal{S}_1$.

▶ **Observation 2.5** (♠). Let $\mathcal{S}$ be the set of all minimal satisfying assignments of size at most k for $\mathcal{F}$. Then for each $S \in \mathcal{S}$, there is $S' \in \mathcal{S}_1$, such that $S' \subseteq S$.

Now we describe the next $k - 1$ passes. The algorithm constructs a set $\mathcal{S}_{\mathsf{prm}}$ of prime partial assignments, which will be enough to resolve the instance. Initially, we set $\mathcal{S}_{\mathsf{prm}} = \mathcal{S}_1$.

Pass $\boldsymbol{\ell}$ ($\mathbf{2 \leq \boldsymbol{\ell} \leq \boldsymbol{k}}$). Consider a non-monotone clause $C = (x_1^C \vee x_2^C \cdots \vee x_{d_1}^C \vee \neg y_1^C \vee \neg y_2^C \vee \ldots \neg y_{d_2}^C)$ (where $d_1 + d_2 \leq d$) seen in the stream. For each $S \in \mathcal{S}_{\mathsf{prm}}$, such that $\{y_1^C, y_2^C, \ldots y_{d_2}^C\} \subseteq S$ and $\{x_1^C, x_2^C, \ldots x_{d_1}^C\} \cap S = \emptyset$ we do the following.

- If $|S| = k$, then remove S from $\mathcal{S}_{\mathsf{prm}}$.
- Otherwise, $|S| \leq k - 1$. Let $\mathcal{S}'_{\mathsf{prm}} = \mathcal{S}_{\mathsf{prm}}$, and for $i \in [d_1]$, let $S_i = S \cup \{x_i^C\}$. Set $\mathcal{S}_{\mathsf{prm}} = (\mathcal{S}'_{\mathsf{prm}} \setminus \{S\}) \cup \{S_i \mid i \in [d_1]\}$.

[1] Proofs of results marked with a ♠ can be found in the appendices.

Clearly, Pass ℓ, where $2 \leq \ell \leq k$, on reading a clause C uses time $\mathrm{O}(|\mathcal{S}_1|dk)$. Moreover, it modifies the sets in $\mathcal{S}_{\textsf{prm}}$ (increasing $|\mathcal{S}_{\textsf{prm}}|$ by at most a factor of d), by either removing a set $S \in \mathcal{S}_1$ completely, or adding one more element to S (when the size is less than k). The above procedure is executed only for $k-1$ passes. Thus, it always maintains that $|\mathcal{S}_{\textsf{prm}}| \in \mathrm{O}\big(d^{\mathrm{O}(k)}\big)$ (see Proposition 2.1) and each set in $\mathcal{S}_{\textsf{prm}}$ has at most k elements (each representable by $\log n$ bits). Thus, the (total) space used by the algorithm is bounded by $\mathrm{O}\big((k^d + d^{\mathrm{O}(k)})k \log n\big)$.

For simplicity of description, we introduce the following notation. We set $\mathcal{S}^1_{\textsf{prm}} = \mathcal{S}_1$ and for each $\ell \in [k]$, we let $\mathcal{S}^{\ell}_{\textsf{prm}}$ denote the the set $\mathcal{S}_{\textsf{prm}}$ after the execution of Pass ℓ. We let $\rho = (Q_1, Q_2, \ldots, Q_t)$ be the sequence of non-monotone clauses in $\mathcal{F}$, where the ordering is given by the order of their appearance in the stream. For $\ell \in [k] \setminus \{1\}$, $i \in [t]$, we let $\mathcal{S}^{\ell}_{\textsf{prm}}(i)$ be the set $\mathcal{S}_{\textsf{prm}}$ (after modification, if any) at Pass ℓ after reading the clause Q_i. Furthermore, we let $\mathcal{S}^{\ell}_{\textsf{prm}}(0)$ be the set $\mathcal{S}^{\ell-1}_{\textsf{prm}}$. Next, we prove some results that will be useful in establishing the correctness of the algorithm.

▶ **Lemma 2.6.** *Let $\mathcal{S}$ be the set of all minimal assignments for $\mathcal{F}$ of size at most k. For each $\ell \in [k]$ and $S \in \mathcal{S}$, there is $S' \in \mathcal{S}^{\ell}_{prm}$, such that $S' \subseteq S$.*

Proof. We prove this using induction on ℓ. The claim follows for $\ell = 1$ from Observation 2.5. This forms the base case of our induction. Next, we assume that the claim holds for each $\ell \leq z$ (for some $1 \leq z \leq k-1$) and then we prove it for $\ell = z+1$. At the beginning of ℓth pass when no non-monotone clause is read from the stream, we have for each $S \in \mathcal{S}$, there is $S' \in \mathcal{S}^{\ell}_{\textsf{prm}}(0)$, such that $S' \subseteq S$. This follows from the fact that $\mathcal{S}^{\ell}_{\textsf{prm}}(0) = \mathcal{S}^{\ell-1}_{\textsf{prm}}$. Next, we assume that at Pass ℓ, the claim holds after reading the clause Q_i, for each $i \leq p$, where $p \in [t-1] \cup \{0\}$. Now we prove the claim for $Q_{p+1} = (x^{p+1}_1 \vee x^{p+1}_2 \cdots \vee x^{p+1}_{d_1} \vee \neg y^{p+1}_1 \vee \neg y^{p+1}_2 \vee \ldots \neg y^{p+1}_{d_2})$. Consider $S \in \mathcal{S}$ and let $\hat{S} \in \mathcal{S}^{\ell}_{\textsf{prm}}(p)$, such that $\hat{S} \subseteq S$. We will show that there is a set $S' \in \mathcal{S}^{\ell}_{\textsf{prm}}(p+1)$, such that $S' \subseteq S$. Let $X = \{x^{p+1}_1, x^{p+1}_2, \ldots, x^{p+1}_{d_1}\}$ and $Y = \{y^{p+1}_1, y^{p+1}_2, \ldots, y^{p+1}_{d_2}\}$. If $Y \not\subseteq \hat{S}$ or $X \cap \hat{S} \neq \emptyset$, then $\hat{S} \in \mathcal{S}^{\ell}_{\textsf{prm}}(p+1)$. Hence, $S' = \hat{S}$ is a set such that $S' \subseteq S$. Otherwise, we have $Y \subseteq \hat{S}$ and $X \cap \hat{S} = \emptyset$. Since S satisfies Q_{p+1}, it must contain a variable, say $x^{p+1}_{i^*}$ from $\{x^{p+1}_1, x^{p+1}_2, \ldots, x^{p+1}_{d_1}\}$. As $X \cap \hat{S} = \emptyset$, $\hat{S} \subseteq S$, $|S| \leq k$, and $x^{p+1}_{i^*} \in S$, we have that $|S| \leq k-1$. For $i \in [d_1]$, let $\hat{S}_i = \hat{S} \cup \{x^{p+1}_i\}$. Recall that $\mathcal{S}^{\ell}_{\textsf{prm}}(p+1) = (\mathcal{S}^{\ell}_{\textsf{prm}}(p) \setminus \{\hat{S}\}) \cup \{\hat{S}_i \mid i \in [d_1]\}$. From the above we can conclude that $\hat{S}_{i^*} \subseteq S$ and $\hat{S}_{i^*} \in \mathcal{S}^{\ell}_{\textsf{prm}}(p+1)$. This concludes the proof. ◀

▶ **Observation 2.7.** For $i \in [k-1]$ and a set $S \in \mathcal{S}^{i}_{\textsf{prm}}$, if $S \in \mathcal{S}^{i+1}_{\textsf{prm}}$, then for each $\ell \in \{i, i+1, \ldots, k\}$, we have $S \in \mathcal{S}^{\ell}_{\textsf{prm}}$.

Proof. Consider $i \in [k-1]$ and a set $S \in \mathcal{S}^{i}_{\textsf{prm}}$, such that $S \in \mathcal{S}^{i+1}_{\textsf{prm}}$. Let $\ell \in \{i+2, i+3 \ldots, k\}$ be the lowest integer, such that $S \notin \mathcal{S}^{\ell}_{\textsf{prm}}$ (if such an ℓ does not exist, the claim trivially holds). Since $S \in \mathcal{S}^{\ell-1}_{\textsf{prm}}$ and $S \notin \mathcal{S}^{\ell}_{\textsf{prm}}$, there is a non-monotone clause $Q = (x_1 \vee x_2 \cdots \vee x_{d_1} \vee \neg y_1 \vee \neg y_2 \vee \ldots \neg y_{d_2})$, such that $\{y_1, y_2, \ldots, y_{d_2}\} \subseteq S$ and $\{x_1, x_2, \ldots, x_{d_1}\} \cap S = \emptyset$. But we also encountered Q at $(\ell-1)$th pass, and S should have been modified/deleted, which is a contradiction. ◀

▶ **Lemma 2.8.** *Let $\mathcal{S}$ be the set of all assignments for $\mathcal{F}$ of size at most k. For every $S \in \mathcal{S}$, there is $S' \in \mathcal{S}_{prm}$, such that $S' \subseteq S$ and S' satisfies every clause of $\mathcal{F}$.*

Proof. Consider $S \in \mathcal{S}$ and let $S' \in \mathcal{S}_{\textsf{prm}} = \mathcal{S}^{k}_{\textsf{prm}}$ be a set such that $S' \subseteq S$. The existence of S' is guaranteed by Lemma 2.6. We will show that S' satisfies all the clauses of $\mathcal{F}$. By the construction of $\mathcal{S}_{\textsf{prm}}$, there is a set $\hat{S} \in \mathcal{S}_1$, such that $\hat{S} \subseteq S'$. Thus, S' satisfies each monotone clause of $\mathcal{F}$ (see Proposition 2.1 and 2.2). Next, consider an anti-monotone clause

$C = (\neg y_1 \vee \neg y_2 \vee \ldots \neg y_{d'})$ (where $d' \leq d$), and let $Y = \{y_1, y_2, \ldots, y_{d'}\}$. Since S satisfies C, $Y_S = Y \setminus S$ is a non-empty set. As $S' \subseteq S$, we have $S' \cap Y_S = \emptyset$. Thus, S' satisfies C. If S' satisfies all the non-monotone clauses of $\mathcal{F}$, then the claim follows. Otherwise, let $C = (x_1 \vee x_2 \cdots \vee x_{d_1} \vee \neg y_1 \vee \neg y_2 \vee \ldots \neg y_{d_2})$ be a non-monotone clause in $\mathcal{F}$ which is not satisfied by S', and let $X = \{x_1, x_2, \ldots, x_{d_1}\}$ and $Y = \{y_1, y_2, \ldots, y_{d_2}\}$. Since S' does not satisfy C, we have $Y \subseteq S'$ and $X \cap S' = \emptyset$. Notice that $Y \subseteq S$ as $S' \subseteq S$. As S satisfies C, we have $S \cap X \neq \emptyset$. This together with the fact that $X \cap S' = \emptyset$ implies that $|S'| \leq k-1$. We can assume that $\hat{S} \neq \emptyset$, as $\mathcal{S}_{\mathsf{prm}}$ can be assumed to contain only non-empty sets, otherwise, $\emptyset$ is a solution to $\mathcal{F}$. The above discussions together with Observation 2.7 and the fact that $|S'| \leq k-1$, implies that $S' \in \mathcal{S}^{k-1}_{\mathsf{prm}}$ (and we have $S' \in \mathcal{S}^{k}_{\mathsf{prm}}$). But then at the kth pass, we would have encountered C, and S' would be replaced by d_1 many sets, namely $S' \cup \{x_i\}$, for each $i \in [d_1]$. This concludes the proof. ◀

In the $(k+1)^{\text{th}}$ pass, the algorithm performs the following steps, whose correctness is established by the discussion above.

Pass $k+1$. Consider a clause C seen in the stream. If there is $S \in \mathcal{S}_{\mathsf{prm}}$, such that S does not satisfy C, then remove S from $\mathcal{S}_{\mathsf{prm}}$. When the stream is over, if $\mathcal{S}_{\mathsf{prm}} \neq \emptyset$, then return yes, and otherwise, return no.

We now have the following theorem.

▶ **Theorem 2.9.** *Instances (F,k) of* Min-Ones *d-SAT ($d \geq 2$) can be solved in $k+1$ passes using space* $\mathrm{O}\big((kd^{ck} + k^d)\log n\big)$ *($c > 0$, a constant).*

By carefully adapting the standard branching algorithm for Min-Ones-d-SAT, we obtain the following theorem.

▶ **Theorem 2.10 (♠).** *Instances (F,k) of* Min-Ones *d-SAT ($d \geq 2$) can be solved in $(d+1)^k$ passes using space $\mathrm{O}(k \log n)$.*

Using Theorem 2.9 and 2.10 we can obtain the following result for IP$_2$, a restricted Integer Programming problem in which every constraint has at most 2 variables (see Appendix D for details).

▶ **Theorem 2.11 (♠).** IP$_2$ *admits algorithms that solve instances (P,k) in*

- *$k+1$ passes using space $\mathrm{O}(f(k)\log n)$ ($f : \mathbb{N} \to \mathbb{N}$, a computable function), and in*
- *3^k passes using space $\mathrm{O}(f(k)\log n)$.*

3 Streaming Kernelizations

In this section, we describe a kernelization for Min-Ones 2-SAT that makes $k+2$ passes over instances $(\mathcal{F},k)$ using space $\mathrm{O}\big(k^6 \log n\big)$ and produces a kernel with $\mathrm{O}\big(k^6\big)$ clauses. In the first pass, the algorithm computes a set of monotone clauses as in Section 2. Then over k more passes, for each variable x appearing in these clauses, the algorithm computes a set of variables which must be set to one if x is set to 1, and the implications that force this. In the last pass, it collects all anti-monotone clauses which only contain variables that also appear in the stored clauses.

We now formally describe our algorithm. Let $(\mathcal{F},k)$ be an instance of Min-Ones 2-SAT on n variables. In the first pass we apply the algorithm Stream-HS of Proposition 2.2 to obtain a set of monotone clauses, $\mathcal{C}_1$. That is, we do the following.

Pass 1. Obtain a set $\mathcal{C}_1$ of monotone clauses of $\mathcal{F}$ using the same procedure as the first pass of Section 2.

Let V be the set of variables appearing in $\mathcal{F}$, V_1 be the set of variables appearing in $\mathcal{C}_1$. For each variable $v \in V_1$, we maintain a set of variables P_v and a set of clauses $\mathcal{P}_v$. Initially, $P_v = \{v\}$ and $\mathcal{P}_v = \emptyset$, for $v \in V_1$. Now we are ready to describe our next k passes.

Pass ℓ. Consider a non-monotone clause $C = (x \vee \neg y)$ seen in the stream. For each $v \in V_1$ such that $y \in P_v$, $x \notin P_v$, $C \notin \mathcal{P}_v$, and $|P_v| \leq k$, add x and C to the sets P_v and $\mathcal{P}_v$, respectively.

For $v \in V_1$ and $\ell \in [k+1]$, by $P_v(\ell)$ we denote the set P_v at the end of pass ℓ (or at the beginning of pass $\ell + 1$, when $\ell = 1$). Furthermore, we let $P = \cup_{v \in V_1} P_v$ and $\mathcal{P} = \cup_{v \in V_1} \mathcal{P}_v$.

▶ **Observation 3.1 (♠).** Let $i \in [k]$ and $v \in V_1$, such that $|P_v(i)| = |P_v(i+1)|$. For all $\ell \in \{i, i+1, \ldots, k+1\}$, we have $|P_v(\ell)| = |P_v(i)|$.

▶ **Lemma 3.2.** *Let S be an assignment which satisfies all clauses in $\mathcal{P}$. For each $v \in V_1 \cap S$, we have $P_v \subseteq S$.*

Proof. Consider $v \in V_1 \cap S$ and let $\rho = (C_1 = (x_1 \vee \neg y_1), C_2 = (x_2 \vee \neg y_2), \ldots, C_t = (x_t \vee \neg y_t))$ be the order in which the clauses in $\mathcal{P}_v$ were added. Note that $P_x = \{x_i \mid i \in [t]\}$. We will show by induction on the index $i \in [t]$ that each $x_i \in S$. Before reading C_1, the only element in P_v was v. As C_1 was added to $\mathcal{P}_v$, it must hold that $y_1 = v$. Since $v \in S$, and S satisfies each clause in $\mathcal{P}$, S must contain x_1. For the induction hypothesis, we suppose that for some $p \in [t-1]$, we have $\{x_i \mid i \in [p]\} \subseteq S$. We will now show that $x_{p+1} \in S$. Since $C_{p+1} \in \mathcal{P}_v$ and C_{p+1} appears after C_i in ρ, for each $i \in [p]$, there exists $z \in \{x_i \mid i \in [p]\}$, such that $z = y_{p+1}$. But since $z \in S$ and S satisfies each clause in $\mathcal{P}$, we have that $x_{p+1} \in S$. ◀

Let $\mathcal{F}'$ be the 2-CNF formula containing all the anti-monotone clauses of $\mathcal{F}$ and all the clauses in $\mathcal{C}_1 \cup \mathcal{P}$.

▶ **Lemma 3.3.** *$(\mathcal{F}, k)$ is a YES instance of* Min-Ones 2-SAT *if and only if $(\mathcal{F}', k)$ is a YES instance of* Min-Ones 2-SAT.

Proof. The forward direction follows from the fact that each clause in $\mathcal{F}'$ is also a clause in $\mathcal{F}$. In the backward direction, let S be a solution to Min-Ones 2-SAT in $(\mathcal{F}', k)$, and $S' = \bigcup_{v \in V_1 \cap S} P_v$. We show that S' is a solution to Min-Ones-2-SAT in $(\mathcal{F}, k)$. Since $V_1 \cap S \subseteq S'$, from Proposition 2.2 we have that S' satisfies each monotone clause of $\mathcal{F}$. From Lemma 3.2 we have $S' \subseteq S$. Thus, S' satisfies each anti-monotone clause of $\mathcal{F}$ ($\mathcal{F}'$ contains all of them). If S' satisfies each non-monotone clause of $\mathcal{F}$, then the claim follows. Otherwise, we have a non-monotone clause $C = (x \vee \neg y)$ in $\mathcal{F}$, which is not satisfied by S'. We have that $x \notin S'$ and $y \in S'$. Let $V_y = \{v \in V_1 \mid y \in P_v\}$. The construction of S' implies that there is $v^* \in V_y$ such that $v^* \in S$. From the construction of S' we have that $x \notin P_{v^*}$. The above discussions together with Observation 3.1 implies that we would have encountered C at a pass $i \leq k$, and we did not add x to P_{v^*}. This means that $|P_{v^*}| \geq k+1$. But this contradicts the fact that S has size at most k (note that from Lemma 3.2 we have $P_{v^*} \subseteq S$). ◀

Let $V_2 = V_1 \cup \left(\bigcup_{v \in V_1} P_v\right)$. We will construct a set $\mathcal{B}$ of anti-monotone clauses. Initially, $\mathcal{B} = \emptyset$. We now describe the $(k+2)^{th}$ pass of our algorithm, which constructs the set $\mathcal{B}$.

Pass $k+2$. For each anti-monotone clause $C = (\neg x \vee \neg y)$ in the stream with $\{x, y\} \subseteq V_2$ and $C \notin \mathcal{B}$, add C to $\mathcal{B}$. Then forget the sets P_v, where $v \in V_1$.

Let $\tilde{\mathcal{F}}$ be the 2-CNF formula obtained from $\mathcal{F}$ by removing all anti-monotone clauses that are not in $\mathcal{B}$.

▶ **Lemma 3.4 (♠).** *$(\mathcal{F}, k)$ is a yes-instance of* Min-Ones-2-SAT *if and only if $(\tilde{\mathcal{F}}, k)$ is a yes-instance of* Min-Ones 2-SAT*.*

Notice that we have stored the sets of clauses $\mathcal{C}_1$, $\mathcal{P}$, and $\mathcal{B}$, of sizes $\mathrm{O}(k^2)$, $\mathrm{O}(k^3)$, and $\mathrm{O}(k^6)$, respectively. This results in the instance $(\tilde{\mathcal{F}}, k)$ of Min-Ones 2-SAT. The above discussions together with Lemma 3.4 implies the following theorem.

▶ **Theorem 3.5.** Min-Ones-2-SAT *admits an algorithm that kernelizes instances (F, k) in $k+2$ passes using space $\mathrm{O}(k^6 \log n)$ and produces a kernel with $\mathrm{O}(k^6)$ clauses.*

4 Lower Bounds

We begin this section by exhibiting a reduction from the POT Pointer Chasing problem (defined later) to Min-Ones 2-SAT and use it to prove the following theorem.

▶ **Theorem 4.1.** *Any streaming algorithm that solves instances (F, k) of* Min-Ones *d-SAT ($d \geq 2$) in k passes requires space $\Omega(\max\{n^{1/k}/2^k, \log \frac{n}{k}\})$, where n is the number of variables in F.*

The well-known *truncated* disjointness problem of Communication Complexity has the following lower bound.

▶ **Proposition 4.2** (Kushilevitz and Nisan [26], Example 2.12)**.** *Let $n, k \in \mathbb{N}$ with $0 \leq k \leq \lfloor n/2 \rfloor$. Any deterministic protocol for DISJ_k requires $\Omega(\log \binom{n}{k})$ bits of communication overall .*

For some background on DISJ_k and other problems (INDEX and DISJ) appearing in the proofs below, the reader is referred to Kushilevitz and Nisan's standard work on Communication Complexity [26].

Using the bound of Proposition 4.2, it is possible to prove the intuitively obvious notion that a streaming algorithm which needs to keep track of locations in its input must use space $\Omega(\log n)$, where n is the size of its input.

▶ **Lemma 4.3.** *Let `MOdSSolve` be a streaming algorithm for* Min-Ones *d-SAT ($d \geq 2$) that solves instances (F, k) of* Min-Ones *d-SAT on n variables using space $g(n, k)$. For any $k \in \{1, \ldots, \lfloor n/2 \rfloor\}$, if `MOdSSolve` makes p passes to solve instances (F, k), then $g(n, k) = \Omega((1/p) \log \binom{n}{k})$.*

Proof. Consider the following protocol for DISJ_k, in which Alice receives the set $S \subseteq \{1, \ldots, n\}$ and Bob receives the set $T \subseteq \{1, \ldots, n\}$ ($|S|, |T| = k$). Alice constructs the forumla $F_S = \bigwedge_{i \in S} \neg x_i \vee \neg x_i$ and Bob constructs the formula $F_T = \bigwedge_{i \in T} x_i \vee x_i$. Observe that $(F_S \wedge F_T, k)$ is a `YES` instance of Min-Ones 2-SAT if and only if $S \cap T = \emptyset$.

Now alice runs `MOdSSolve` with parameter k and F_S as partial input, and passes its memory r_S to Bob. Bob resumes execution of `MOdSSolve` on the memory r_S and feed it the formula F_T. With this, the algorithm makes the first pass over $F_S \wedge F_T$. Bob then passes the algorithm's memory r_T back to Alice. Using r_T, Alice resumes execution of `MOdSSolve`. The process is repeated for as many passes as the algorithm requires over $F_S \wedge F_T$. Once the algorithm halts, Bob returns its output as his answer.

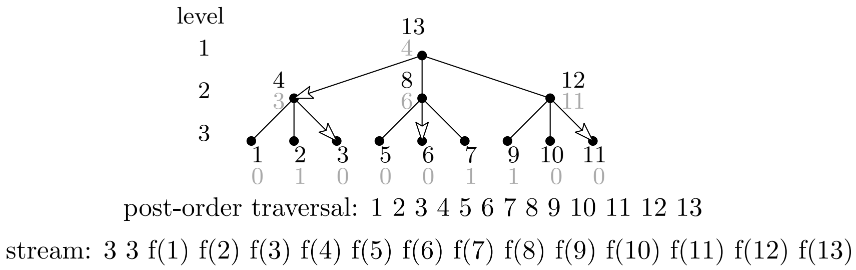

level
1
2
3
13
4
4
3
8
6
12
11
1 2 3 5 6 7 9 10 11
0 1 0 0 0 1 1 0 0
post-order traversal: 1 2 3 4 5 6 7 8 9 10 11 12 13
stream: 3 3 f(1) f(2) f(3) f(4) f(5) f(6) f(7) f(8) f(9) f(10) f(11) f(12) f(13)

Figure 1 An instance of POT Pointer Chasing with parameters t = 3 and l = 2. The stream consists of t, k and the values of f appearing as in the *lexicographic* post-order traversal of the tree. In the tree, labels appear in black next to vertices, and the corresponding values of f appear in grey. The chain of pointers leads to the vertex labelled 3, with $f(3) = 0$.

Since `MOdSSolve` outputs `YES` if and only if $(F_S \wedge F_T, k)$ is a `YES` instance, the protocol is valid. The amount of communication per pass between Alice and Bob is at most $2g(n,k)$, so the total amount of communication is at most $2pg(n,k)$. From Proposition 4.2, we have $2pg(n,k) = \Omega\big(\log\binom{n}{k}\big)$, i.e. $g(n,k) = \Omega\big((1/p)\log\binom{n}{k}\big)$. ◀

The above result shows an $\Omega(\log n)$ lower bound on the space used by any algorithm that solves instances (F,k) of Min-Ones d-SAT in $\Omega(k)$ passes. This is quite weak, but it is possible to strengthen the result substantially using a lower bound for the following POT Pointer Chasing problem.

Consider a complete t-ary tree T with $l+1$ levels rooted at the vertex r. Let the levels be numbered from 1 to $l+1$, with the root being on level 1. For each non-leaf vertex v, define v_i to be the i^{th} child of v (in the lexicographic ordering of its children). Given a function $f : \mathrm{V}(T) \to \{0,\dots,t-1\}$, define $f^*(v) = v_{f(v)}$ for non-leaf vertices v and $f^*(v) = f(v)$ for leaf vertices. For $i \in \mathbb{N}$, $(f^*)^i(r)$ denotes the result of applying f^* to r repeatedly, i times.

POT Pointer Chasing

Instance: (T,f), where T is a complete t-ary tree with $l+1$ levels rooted at r, encoded as a post-order traversal of its vertices, and $f : \mathrm{V}(T) \to \{0,\dots,t-1\}$.
Question: Is $(f^*)^l(r) = 1$?

Figure 1 shows an instance with parameters $t = 3$ and $l = 2$. The following result exhibits a tradeoff between the number of passes made by a streaming algorithm for POT Pointer Chasing and the space it requires.

▶ **Proposition 4.4** (Guha and McGregor [18], Theorem 1). *Any p-pass streaming algorithm that solves* POT Pointer Chasing *instances over t-ary trees with $(p+1)$ levels requires space $\Omega(t/2^p)$ in the worst case.*

▶ **Lemma 4.5.** *Let (T,f) be an instance of* POT Pointer Chasing*, where T is a t-ary tree with $k+1$ levels. A boolean formula F can be constructed such that (T,f) is a YES instance of* POT Pointer Chasing *if and only if (F,k) is a YES instance of* Min-Ones 2-SAT*.*

Proof. The tree T has levels $1,\dots,k+1$, with the root r on level 1 and the leaves on level $k+1$. Since each internal vertex has t children, $|V(T)| = \frac{t^{k+1}-1}{t-1} = \mathrm{O}\big(t^k\big)$. Consider the following boolean formula F with $n = \frac{t^k-1}{t-1} = \Theta\big(t^{k-1}\big)$ variables.

Let $w = f^*(r)$, i.e. the $f(r)^{\text{th}}$ child of r, and T_w be the subtree of T rooted at w. The variable set of F is $\{x_v \mid v \in \mathrm{V}(T_w)\}$. For each vertex v on level $i = 2, \ldots, k$ of T, F has the clause $x_v \to x_{f^*(v)} \equiv \neg x_v \vee x_{f^*(v)}$. For each leaf vertex v, F has the clause $\neg x_v \vee \neg x_v$ if and only if $f(v) = 0$. In addition, F has the clause $x_w \vee x_w$.

We now show that (F, k) is an equivalent instance of Min-Ones 2-SAT. Consider the leaf vertex $z = (f^*)^k(r)$, i.e. the vertex reached by chasing pointers from the root of T. If (T, f) is a YES-instance, i.e. $f(z) = 1$, then F can be satisfied by setting k variables (corresponding to variables on the w–z path in T) to 1, i.e. (F, k) is a YES instance. In the other case, i.e. $f(z) = 0$, F is unsatisfiable: F contains the clause $x_w \vee x_w$, a chain of implications from w to z, and the clause $\neg x_z \vee \neg x_z$, which cannot be satisfied simultaneously. Thus, (F, k) is a NO instance. ◀

Observe that the implication $x_v \to x_{f(v)}$ can be produced by simply reading off the value $f(v)$. This is because in the stream, the values of f appear as in the (lexicographic) post-order traversal of T, and knowing the value $f(v)$ and the position of $f(v)$ in the stream is enough to determine the $f(v)^{\text{th}}$ child of v. Thus, the clauses can be produced *on the fly* while making a pass over the post order traversal of T.

We now prove Theorem 4.1.

Proof. Let MOdSSolve be a k-pass streaming algorithm for Min-Ones 2-SAT that uses space $g(n, k)$ on inputs (F, k) over n variables. Consider an algorithm that takes as input an instances (T, f) of POT Pointer Chasing over trees with $k+1$ levels, producing instances (F, k) (over $n = \Theta(t^{k-1})$ variables) of Min-Ones 2-SAT on the fly as above, and feeding them as input to MOdSSolve. Because of Lemma 4.5, the output of A on (F, k) correctly decides (T, f).

The algorithm makes k passes over its input and the amount of space used overall is $\mathrm{O}(g(n, k) + \log n)$. This value is $\Omega(t/2^k)$, by Proposition 4.4. Since $n = \Theta(t^k)$, we have $g(n, k) + \log n = \Omega(n^{1/k}/2^k)$. Consider the case $k \geq \sqrt{\log n}$. The expression $n^{1/k}/2^k$ is $\mathrm{o}(1)$, so $g(n, k) = \Omega(n^{1/k}/2^k)$ holds trivially. In the other case, i.e. $k < \sqrt{\log n}$, we have $g(n, k) = \Omega(\log n)$ by Lemma 4.3, so $g(n, k) + \log n = \mathrm{O}(g(n, k))$, i.e. $g(n, k) = \Omega(n^{1/k}/2^k)$.

Observe that the bound $g(n, k) = \Omega(\log \frac{n}{k})$ holds for any $k \leq \lfloor n/2 \rfloor$ (Lemma 4.3), and for $k > \lfloor n/2 \rfloor$, $g(n, k) = \Omega(\log \frac{n}{k})$ holds trivially. Therefore, we have $g(n, k) = \Omega(\max\{n^{1/k}/2^k, \log \frac{n}{k}\})$. ◀

Suppose a streaming algorithm for Min-Ones 2-SAT uses space $\mathrm{O}(f(k)n^{1/k-\epsilon})$ ($\epsilon > 0$, a constant) to decide instances (F, k) over n variables. Observe that $\lim_{n\to\infty} \frac{f(k)n^{1/k-\epsilon}}{n^{1/k}/2^k} = 0$ for any function f. Thus, we have the following corollary.

▶ **Corollary 4.6.** *Let $\epsilon > 0$ be a number. Any streaming algorithm for* Min-Ones 2-SAT *that uses space $\mathrm{O}(f(k)n^{1/k-\epsilon})$ must make at least $k+1$ passes over its input.*

The preceding corollary shows that the algorithm of Theorem 2.9, which makes $k+1$ passes over (F, k), is the best possible inasmuch as the number of passes is concerned. We now exhibit two lower bounds on the space complexity of Min-Ones 2-SAT using Communication Complexity similar to those in Lemma 4.3, which apply to Min-Ones d-SAT even when $d = 1$.

▶ **Theorem 4.7.** *There are no 1-pass streaming algorithms for* Min-Ones d-SAT *($d \geq 1$) that use space $f(k)g(n)$ ($f, g : \mathbb{N} \to \mathbb{N}$, computable functions; $g = \mathrm{o}(n)$) on instances (F, k) with n variables.*

Proof. Observe that any instance (a, b) of INDEX can be encoded as the formula $F = \left(\bigwedge_{a[i]=1} \neg x_i\right) \wedge (x_b)$. $(F, 1)$ is a `NO` instance if and only if $a[b] = 1$. Suppose there is a 1-pass algorithm for Min-Ones d-SAT that uses space $f(k)g(n)$ on n-variable inputs with parameter k. Alice runs the algorithm on $\bigwedge_{a[i]=1} \neg x_i$ and passes the algorithm's memory to Bob. Bob resumes executing the algorithm on the memory and feeds it the additional clause x_b. Using the output of the algorithm, Bob can determine the value $a[b]$.

It is known that any deterministic 1-pass protocol for INDEX requires $\Omega(n)$ bits of communication (Kushilevitz and Nisan [26], Example 4.19). Because Alice passes the algorithm's memory to Bob, the size of this memory must be $\Omega(n)$, i.e. $f(1)g(n) = \Omega(n)$. Thus, there are no 1-pass parameterized streaming algorithms for Min-Ones d-SAT $(d \geq 1)$ that use space $\mathrm{O}(f(k)g(n))$ with $g = \mathrm{o}(n)$. ◀

The above theorem shows that even in the case where every clause consists of exactly one literal, it is not possible to solve an instance of Min-Ones d-SAT in a single pass without using space $\Omega(n)$. Unlike Theorem 4.1, the next result holds in cases where p, the number of passes made by the algorithm, is a more general function of k.

▶ **Theorem 4.8.** *Any p-pass streaming algorithm for* Min-Ones d-SAT *$(d \geq 1)$ requires space $\Omega(n/p)$.*

Proof. The claim follows from the fact that instances of DISJ can be encoded as SAT formulas in which every clause comprises one literal. Consider the formula $F = \bigwedge(C_S \cup C_T)$, where $C_S = \{x_i \mid i \in S\}$ and $C_T = \{\neg x_i \mid i \in T\}$. $S \cap T = \emptyset$ if and only if F is satisfiable. By standard arguments from Communication Complexity, any p-pass streaming algorithm for Min-Ones 2-SAT must use space $\Omega(n/p)$. ◀

5 Conclusion

In this work, we have proved a variety of results that together provide a complete picture of the parameterized streaming complexity of Min-Ones d-SAT. One of the main results is the streaming algorithm for Min-Ones d-SAT which solves instances (F, k) in $(k+1)$ passes using space $\mathrm{O}\big((kd^{ck} + k^d) \log n\big)$ $(c > 0$, a constant). The matching $(k+1)$-pass lower bound shows that in terms of the number of passes, this result is the best possible.

It is pertinent to note that such results, i.e. which show a sharp tradeoff between the space complexity of a parameterized streaming problem and the number of passes allowed, are quite scarce in the literature. It would be interesting to see which other parameterized streaming problems exhibit such behaviour.

References

1 Faisal N. Abu-Khzam. Kernelization Algorithms for D-Hitting Set Problems. In *Algorithms and Data Structures*, pages 434–445. Springer Berlin Heidelberg, 2007.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *Journal of Computer and System Sciences*, 58(1):137–147, February 1999.

3 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

4 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters*, 8(3):121–123, 1979.

5 Rajesh Chitnis, Graham Cormode, Mohammadtaghi Hajiaghayi, and Morteza Monemizadeh. Parameterized Streaming: Maximal Matching and Vertex Cover. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1234–1251, 2015.

6 Stephen A. Cook. The Complexity of Theorem-proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

7 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

8 Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized complexity.* Springer-Verlag, 2013.

9 Dingzhu Du, Jun Gu, Panos M Pardalos, et al. *Satisfiability problem: theory and applications: DIMACS Workshop, March 11-13, 1996*, volume 35. American Mathematical Soc., 1997.

10 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

11 Stefan Fafianie and Stefan Kratsch. Streaming Kernelization. In *Mathematical Foundations of Computer Science (MFCS)*, pages 275–286, 2014.

12 Philippe Flajolet and G Nigel Martin. Probabilistic counting. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 76–82, 1983.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

14 Fedor V. Fomin and Petteri Kaski. Exact Exponential Algorithms. *Commun. ACM*, 56(3):80–88, March 2013.

15 Lance Fortnow. The Status of the P Versus NP Problem. *Commun. ACM*, 52(9):78–86, September 2009.

16 Weiwei Gong and Xu Zhou. A survey of SAT solver. In *AIP Conference Proceedings*, volume 1836, 2017.

17 Jun Gu, Paul W Purdom, John Franco, and Benjamin W Wah. Algorithms for the satisfiability (sat) problem. In *Handbook of Combinatorial Optimization*, pages 379–572. Springer, 1999.

18 Sudipto Guha and Andrew McGregor. Tight Lower Bounds for Multi-Pass Stream Computation Via Pass Elimination. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5125, pages 760–772, 2008.

19 Dan Gusfield and Leonard Pitt. A Bounded Approximation for the Minimum Cost 2-Sat Problem. *Algorithmica*, 8(1-6):103–117, 1992.

20 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In *External Memory Algorithms, Proceedings of a DIMACS Workshop*, pages 107–118, 1998.

21 Dorit S Hochbaum, Nimrod Megiddo, Joseph (Seffi) Naor, and Arie Tamir. Tight Bounds and 2-Approximation Algorithms for Integer Programs with Two Variables per Inequality. *Mathematical Programming*, 62(1-3):69–83, 1993.

22 David S Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

23 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

24 Stefan Kratsch and Magnus Wahlström. Two Edge Modification Problems without Polynomial Kernels. In *Parameterized and Exact Computation, 4th International Workshop, (IWPEC)*, pages 264–275, 2009.

25 Melven R Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967.

26 Eyal Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, New York, NY, USA, 1997.

27 Neeldhara Misra, N. S. Narayanaswamy, Venkatesh Raman, and Bal Sri Shankar. Solving Min Ones 2-Sat as Fast as Vertex Cover. *Theoretical Computer Science*, 506:115–121, 2013.

28 J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 253–258, 1978.

29 Rolf Niedermeier. *Invitation to fixed-parameter algorithms.* Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

30 Thomas Stützle, Holger Hoos, and Andrea Roli. A review of the literature on local search algorithms for MAX-SAT. *Rapport technique AIDA-01-02, Intellectics Group, Darmstadt University of Technology, Germany*, 2001.

31 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 188–202. Springer, 2003.

32 Gerhard J Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 185–207. Springer, 2003.

33 Mihalis Yannakakis. Node- and Edge-Deletion NP-Complete Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978.

34 Mihalis Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17(3):475–502, 1994.

A A Brief Introduction to Parameterized Complexity

A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x, k), where x is a classical problem instance and k is an integer, which is called the *parameter*. The framework of parameterized complexity was originally introduced to deal with NP-hard problems, with the aim to limit the exponential growth in the running time expression to the parameter alone. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a parameterized problem Π, there is an algorithm that given an instance (x, k), decides whether or not (x, k) is a YES instance of Π in time $f(k) \cdot p(|x|)$, where f is a computable function of k and p is a polynomial in the input size. Another central notion in parameterized complexity is *kernelization*, which mathematically captures the efficiency of a data preprocessing. A typical goal of a kernelization algorithm is to store only "small" amount of information, which is enough to recover the answer to the original instance. The "smallness" of the stored information is quantified by the input parameter. Formally, a *kernelization algorithm* or a *kernel* for a parameterized problem Π is given an input (x, k), and the goal is to obtain an equivalent instance (x', k') of Π in polynomial time, such that $|x'| + k' \leq g(k)$. Here, g is some computable function whose value only depends only on k, and depending on whether it is a linear, polynomial, or exponential function, the kernel is called a linear, polynomial, or exponential kernel, respectively. It is well known that a parameterized problem is FPT if and only if it admits a kernel. Thus, in the literature, the term "kernel" is used for polynomial kernels (unless stated otherwise). For more details on parameterized complexity, we refer the reader to the books of Downey and Fellows [8], Flum and Grohe [13], Niedermeier [29], and the recent book by Cygan et al. [7].

B Missing Proofs from Section 2

Proof of Proposition 2.1

The algorithm Enum-d-HS is given in Algorithm 1. We start by proving the correctness of the algorithm by induction on k. When $k \leq 0$, then the algorithm correctly computes the set $\mathcal{S}_k$ (see Steps 1-6). Let us assume that the algorithm returns the correct output for all $k \leq t$, where $t \in \mathbb{N}$. We will now prove that the output of the algorithm is correct for

Algorithm 1 Enum-d-HS.

```
   Input: A set $\mathcal{X}$, of subsets of size at most $d$ of a universe $U$, and an integer $k$.
   Output: The (multi)set $\mathcal{S}_k$, of all minimal $d$-hitting sets of size at most $k$.
 1 if $k < 0$ or $\emptyset \in \mathcal{X}$ then
 2     return $\emptyset$;                            /* no hitting set possible */
 3 if $k = 0$ and there is a non-empty set $F \in \mathcal{X}$ then
 4     return $\emptyset$;                            /* no hitting set possible */
 5 if $k = 0$ or there is no set in $\mathcal{X}$ then
 6     return $\{\emptyset\}$;                        /* $\emptyset$ is a hitting set for $\emptyset$ */
 7 Set $\mathcal{S}_k = \emptyset$;
 8 Let $X = \{x_1, x_2, \dots, x_{d'}\}$ (where $d' \leq d$) be an arbitrary non-empty set in $\mathcal{X}$;
 9 for $i = 1$ to $d'$ do
10     Let $\mathcal{X}_i = \{Y \in \mathcal{X} \mid x_i \notin Y\}$;
11     $\mathcal{S}^i$ =Enum-$d$-HS($\mathcal{X}_i, U \setminus \{x_i\}, k-1$);
12     for each $S \in \mathcal{S}^i$ do
13         $\mathcal{S}_k = \mathcal{S}_k \cup \{S \cup \{x_i\}\}$;
14 Remove those sets from $\mathcal{S}_k$ which are not minimal solutions to $(\mathcal{X}, U, k)$;
15 return $\mathcal{S}_k$;
```

$k = t + 1 \geq 1$. If there is no non-empty set in $\mathcal{X}$, then the algorithm returns the correct output (Steps 1-2 and 5-6). Hereafter, we assume that Steps 1-6 are not executed (otherwise, we already have the correct output). Also, we have that $k \geq 1$ and there is a non-empty set $X = \{x_1, x_2, \dots, x_{d'}\} \in \mathcal{X}$. Any d-hitting set must contain at least one element from X. By induction hypothesis, for each $i \in [d']$, we (correctly) compute the set $\mathcal{S}^i$ of all minimal d-hitting sets of size at most $k-1$, for the instance $(\mathcal{X}_i, U \setminus \{x_i\}, k-1)$. Notice that each set $S \in \mathcal{S}^i$, intersects each set in $\mathcal{X}_i$ and may not intersect X. Moreover, $S \cup \{x_i\}$ is a d-hitting set for $(\mathcal{X}, U, k)$. From the above discussion (together with the induction hypothesis), we obtain that $\mathcal{S}^i_k = \{S \cup \{x_i\} \mid S \in \mathcal{S}^i\}$ is a set containing all minimal d-hitting sets containing x_i for $(\mathcal{X}, U, k)$. Thus, $\cup_{i \in [d']} \mathcal{S}^i_k$ is a set containing all minimal d-hitting sets for $(\mathcal{X}, U, k)$. Moreover, by construction we have that $\mathcal{S}_k = \cup_{i \in [d']} \mathcal{S}^i_k$ with non-minimal solutions removed, is the output returned by the algorithm at Step 17. This concludes the proof of correctness of the algorithm.

We now move to the running time analysis of the algorithm. Notice that the running time of the algorithm is given by the recurrence: $T(k) = d \cdot T(k-1) + \mathrm{O}(|U| + |\mathcal{X}| + |\mathcal{S}_k|)$. Also, the size of $\mathcal{S}_k$ is given by the recurrence $D(k) = d \cdot D(k-1)$, where $0 \leq D(0) \leq 1$. Thus, the running time of the algorithm is bounded by $\mathrm{O}(d^k \|I\|)$ and $|\mathcal{S}_k| \in \mathrm{O}(d^k)$. Next, we move to the analysis of the space used by the algorithm. Notice that at any point of time, in the recursive procedure, memory is allocated for at most k copies of Enum-d-HS. Hence, the space required by the algorithm can be bounded by $\mathrm{O}(k\|I\| + kd^k b_U)$. ◀

Proof of Lemma 2.3

From Proposition 2.2, Pass 1 can compute $I_t = (\mathcal{X}_t, U_t, k)$ after reading all the clauses from the stream using $\mathrm{O}(k^d \log n)$ space, and using $\mathrm{O}(k^d)$ time after reading a clause from the stream. Furthermore, $|\mathcal{X}_t| \in \mathrm{O}(k^d)$, and elements of U_t are represented using $\log n$ bits (by Proposition 2.2 and our assumption that variables of $\mathcal{F}$ are $x_1, x_2, \dots, x_n$). Now using Enum-d-HS of Proposition 2.1, the algorithm computes $\mathcal{S}_1$ using space (in bits) bounded by $\mathrm{O}((k^d + d^k)k \log n)$ and time bounded by $\mathrm{O}(d^k k^d \log n)$. ◀

Proof of Observation 2.5

Any minimal satisfying assignment $S \in \mathcal{S}$ is also a satisfying assignment for $\mathcal{F}^+$. From Observation 2.4 we know that $\mathcal{S}_1$ is the set of all minimal satisfying assignments of size at most k for $\mathcal{F}^+$. Hence, it follows that there is $S' \in \mathcal{S}_1$, such that $S' \subseteq S$. ◀

B.1 $(\mathrm{O}(d^k), \mathrm{O}(k))$-streaming-FPT Algorithm for Min-Ones-d-SAT

In this section, we design an $(\mathrm{O}(d^k), \mathrm{O}(k))$-streaming-FPT algorithm for Min-Ones-d-SAT. The algorithm closely follows the standard $\mathrm{O}(d^k)(n+m)^{\mathrm{O}(1)}$ branching algorithm for Min-Ones-d-SAT, where n and m are the number of variables and clauses in the input instance.

Let $(\mathcal{F}, k)$ be an instance of Min-Ones-d-SAT. By $\mathbb{S}$, we denote the stream of clauses in $\mathcal{F}$. We give our $(\mathrm{O}(d^k), \mathrm{O}(k))$-streaming-FPT algorithm Stream-MOS, for Min-Ones-d-SAT algorithm in Algorithm 2. In the following, we describe various functions of the algorithm Stream-MOS. We note that each of the functions have access to the stream $\mathbb{S}$ and a global variable called pass-count.

1. The function `FinishScan` takes no input and returns no output (only updates pass-count). Its goal is only to read the stream till the end and update pass-count, which stores the number of passes we have made through $\mathbb{S}$. When we enter this function, the pass number is updated. If we are already at the end of the stream $\mathbb{S}$, then it exits without doing any other operation. Otherwise, it read $\mathbb{S}$ till the end and exits. The purpose of defining this function (and maintaining pass-count) is to simplify the analysis of the algorithm.
2. The function `TestSatisfiability` takes as input a set S, and its objective is to determine whether or not S satisfies each clause of $\mathcal{F}$. A call to `TestSatisfiability`, makes a complete scan through $\mathbb{S}$ and we explicitly ensure that whenever it is called, we are at the beginning of the stream. Whenever we find a clause unsatisfied by S in the stream, the function calls `FinishScan` to complete the scanning through remaining clauses of $\mathbb{S}$ and update pass-count, and then it exits after returning 0. In the case when there is no clause which is not satisfied by S, it makes a call to `FinishScan` to update pass-count, and exits after returning 1.
3. The function `FindBranchClause` takes as input a set S. Its objective is to find a clause C which cannot be satisfied (by just) setting variables in S to 1. More precisely, it returns a clause C (if it exists) which satisfies two conditions (to be stated, shortly). Let X and Y be the sets of variables which appear positively and negatively in C, respectively. It must hold that $Y \subseteq S$ and $X \cap S = \emptyset$. Notice that for a satisfying assignment S' for $\mathcal{F}$, such that $S \subseteq S'$, it must hold that $S' \cap X \neq \emptyset$. Moreover, as $S \cap X = \emptyset$, S' must contain at least one more vertex (from X), which is not present in S. We will later see how we use C to progress our branching procedure. To find C, `FindBranchClause` makes a complete scan through $\mathbb{S}$. If it finds a clause C with the desired properties, it makes a call to `FinishScan` to complete the scan through $\mathbb{S}$ and update pass-count, and then it exits after returning C. If a clause with the desired properties is not found even when we reach the end of the stream $\mathbb{S}$, it makes a call to `FinishScan` to update pass-count, and then exits after returning $\diamondsuit$ (indicating that a clause with the desired property could not be found).
4. The function `DetectSolution` takes as input a set S, and its objective is to determine whether or not there is a solution for $(\mathcal{F}, k)$ which sets each variable in S to 1. This function is defined because our algorithm is a recursive procedure, and as the algorithm progresses, we maintain a set of variables that have already been set to 1. We note that at

Algorithm 2 Algorithm Stream-MOS.

```
Input: A stream of clauses $\mathbb{S}$ for an instance $(\mathcal{F}, k)$ of MIN-ONES-$d$-SAT.
pass-count=0;
Function FinishScan()
    pass-count = pass-count+1;
    if at end of the stream $\mathbb{S}$ then
        return;
    while end of the stream $\mathbb{S}$ is not reached do
        Read the next clause in the stream;
    return;
Function TestSatisfiability(Set $S$)
    while end of the stream $\mathbb{S}$ is not reached do
        Read the next clause $C$ in the stream;
        if $C$ is not satisfied by $S$ then
            FinishScan();
            return 0;
    FinishScan();
    return 1;
Function FindBranchClause(Set $S$)
    while end of the stream $\mathbb{S}$ is not reached do
        Read the next clause $C$ in the stream, and let $X$ and $Y$ be the sets of
          variables in $C$ appearing positively and negatively, respectively;
        if $Y \subseteq S$ and $S \cap X = \emptyset$ then
            FinishScan();
            return $C$;
    return $\diamondsuit$;
Function DetectSolution(Set $S$)
    if $S > k$ then
        return 0;
    if TestSatisfiability($S$)= 1 then
        return 1;
    $C$ = FindBranchClause($S$);
    if $C \neq \diamondsuit$ then
        if $|S| = k$ then
            return 0;
        Let $X = \{x_1, x_2, \ldots, x_{d'}\}$ (where $d' \leq d$) be the set of variables appearing
          positively in $C$;
        ans = 0;
        for $i = 1$ to $d'$ do
            ans = ans $\vee$ DetectSolution($S \cup \{x_i\}$);
        return ans;
    return 0;
Function MainMOS()
    res = DetectSolution($\emptyset$);
    return res;
```

any point of time we allocate memory only for one such set, and whenever we make calls to other functions, we send the memory location, instead of a separate copy of the set itself. At some steps we call other functions with a modified set (with an element added to S), in this case also we send the memory address after appending the new element (in the front). The above can be achieved by using appropriate memory pointers. Next, we describe the working of `DetectSolution`. If $|S| > k$, then it (correctly) return 0, indicating that there is no satisfying assignment of size at most k containing S. Hereafter, we assume that $|S| \leq k$. Now the function checks if S is a satisfying assignment for $\mathcal{F}$, by making a call to `TestSatisfiability` with (memory location of) S as the argument. If `TestSatisfiability`(S) returns 1, then the function exits after (correctly) returning 1. Otherwise, it makes a call to `FindBranchClause` with (memory location of) S as the argument, and stores the output of it in C. Next, it considers the case when $C \neq \diamondsuit$. Let X and Y be the sets of variables appearing positively and negatively in C, respectively. By the properties of the clauses returned by `FindBranchClause`, we know that $X \cap S = \emptyset$ and $Y \subseteq S$. Thus, for any satisfying assignment S' for $\mathcal{F}$ with $S \subseteq S'$, $S' \cap X \neq \emptyset$ must hold. As $X \cap S = \emptyset$, S' must contain at least one vertex from X and this vertex does not belong to S. If $|S| = k$, then there cannot be a satisfying assignment of size at most k containing S, as otherwise, it will not satisfy C. Thus, in the above case, the function correctly returns 0, and exits. Next, the function deals with the case when $|S| < k$. For any $x \in X$, it checks if there is a satisfying assignment for $\mathcal{F}$ of size at most k containing $S \cup \{x\}$. This is done by making a recursive call to `DetectSolution` with (the memory location of) $S \cup \{x\}$ as the argument. If for any $x \in X$, `DetectSolution`$(S \cup \{x\})$ returns 1, then the function exits after (correctly) returning 1. If for no $x \in X$, `DetectSolution`$(S \cup \{x\})$ returns 1, then the function exits after (correctly) returning 0. If none of the above statements could be used to return an answer, then the algorithm returns 0 and exits.

5. The function `MainMOS` is the main function of the algorithm, where the algorithm begins its execution. The objective of `MainMOS` is to return 1 if $(\mathcal{F}, k)$ is a yes-instance of Min-Ones-d-SAT and return 0, otherwise. Thus, we have only statement, namely, `DetectSolution`$(\emptyset)$ in this function. The correctness of this function follows from the correctness of `DetectSolution`.

Next, we state a lemma regarding Stream-MOS, which will be used to establish the main theorem of this section.

▶ **Lemma B.1.** *Stream-MOS correctly resolves an instance* Min-Ones-d-SAT *(presented as a stream $\mathbb{S}$, of clauses). Moreover, it uses space bounded by* $\mathrm{O}(k \log n)$ *and makes at most* $\mathrm{O}(d^k)$ *passes over* $\mathbb{S}$.

Proof. The correctness of Stream-MOS is immediate from the correctness of each of its functions (which is apparent from their respective descriptions). We now bound the space used by the algorithm and the number of passes it makes over $\mathbb{S}$. The space bounds follows from the facts that at any point of the time, we have at most $\mathrm{O}(k)$ active instances of `DetectSolution` and whenever we pass a set as an argument to a function, its memory is passed, rather than a copy of the set itself. To bound the number of passes that the algorithm makes over $\mathbb{S}$, it is enough to bound pass-count. Recall that pass-count is updated only when `TestSatisfiability` or `FindBranchClause` is called by `DetectSolution`. In the above, the pass-count is updated by `TestSatisfiability` or `FindBranchClause` by making a call to `FinishScan`, which increments pass-count exactly by 1. Observe that the total number of (recursive) calls to `TestSatisfiability` or `FindBranchClause`, made by `DetectSolution` is bounded by $\mathrm{O}(d^k)$. Thus, pass-count is bounded by $\mathrm{O}(d^k)$. This concludes the proof. ◀

The proof of Theorem 2.10 follows from Lemma B.1.

C Missing Proofs from Section 3

Proof of Observation 3.1

Consider $i \in [k]$ and $v \in V_1$, such that $|P_v(i)| = |P_v(i+1)|$. Let $\ell \in \{i+2, i+3 \dots, k+1\}$ be the lowest integer such that $|P_v(\ell)| \neq |P_v(i)|$ (if such an ℓ does not exist, the claim trivially holds). Since $|P_v(\ell-1)| = |P_v(i)|$ and $|P_v(\ell)| \neq |P_v(i)|$, there is a non-monotone clause $Q = (x \vee \neg y)$, such that $y \in P_v(\ell-1)$ and $x \notin P_v(\ell-1)$. But we also encountered C in pass $(\ell-1)$, and P_v should have been modified, which is a contradiction. ◀

Proof of Lemma 3.4

From Lemma 3.3, it is enough to show that $(\mathcal{F}', k)$ is a yes-instance of Min-Ones-2-SAT if and only if $(\tilde{\mathcal{F}}, k)$ is a yes-instance of Min-Ones 2-SAT.

The forward direction follows from the fact that each clause in $\mathcal{F}'$ is also a clause in $\tilde{\mathcal{F}}$. In the backward direction, let S be a solution to Min-Ones 2-SAT in $(\tilde{\mathcal{F}}, k)$. Notice that S satisfies all monotone and non-monotone clauses of $\mathcal{F}'$. For an anti-monotone clause $C = (\neg x \vee \neg y)$, if at least one of x or y is not in V_2, say $x \notin V_2$, then $x \notin S$ (since $S \subseteq V_2$). Otherwise, $x, y \in V_2$, and then C is also a clause in $\tilde{\mathcal{F}}$. Thus, C is satisfied by S. ◀

D Streaming FPT Algorithm for IP$_2$

In this section, we consider a restriction of the integer programming problem, IP$_2$ (defined below). We show how to convert an instance of IP$_2$ to an instance of Min-Ones 2-SAT under parameterized streaming constraints, using the approach of Hochbaum et al. [21]. This allows us to use the algorithms for Min-Ones 2-SAT to solve IP$_2$. We consider integer programs on n variables and m constraints that have the following form.

$$\begin{aligned}
&\text{Minimize } \sum_{j=1}^{n} w_j x_j, \text{ subject to} \\
&\qquad a_i x_{p_i} + b_i x_{q_i} \geq c_i, && (i \in [m], p_i, q_i \in [n]), \\
&\qquad 0 \leq x_j \leq u_j, && (j \in [n]), \text{ and} \\
&\qquad x_j \in \{0, 1\}, && (j \in [n]).
\end{aligned}$$

where the coefficients appearing in the constraints are integers, and for all $j \in [n]$, $w_j \in \mathbb{N}$.

Such integer programs (hereafter called *bounded* integer programs) were considered by Hochbaum et al. [21], who showed that by applying a transformation to the variables of the program, the problem of finding a feasible solution becomes equivalent to 2-SAT. We consider the following problem.

IP$_2$
Input: A bounded-IP $\mathcal{P}$, where we want to minimize $\sum_{j=1}^{n} w_j x_j$, subject to $a_i x_{p_i} + b_i x_{q_i} \geq c_i$, for $i \in [m]$ and $0 \leq x_j \leq u_j$, for $j \in [n]$, and an integer $k \in \mathbb{N}$.
Question: Is there a is feasible solution for $\mathcal{P}$, such that $\sum_{j=1}^{n} w_j x_j \leq k$?

Let $(\mathcal{P}, k)$ be an instance of bounded-IP, where $\mathcal{P}$ is provided as a stream of w_i, for $i \in [n]$, followed by the constraints. As a constraint arrives, we show how we create 2-CNF clauses for it. This will give us an instance of $(\mathcal{F}, k)$, such that $(\mathcal{P}, k)$ is a yes-instance of IP$_2$ if and only if $(\mathcal{F}, k)$ is a yes-instance of Min-Ones-2-SAT. We note that both the construction and the equivalence of the instances follows from [21], therefore, we only briefly explain the construction of $\mathcal{F}$.

We use the approach described in Section 4 of [21] to construct $\mathcal{F}$. Consider the variable constraint $0 \leq x_p \leq u_p$, for $p \in [n]$. By replacing x_p with u_p binary variables $x_{p,l}$ ($l \in [u_p]$) and introducing the constraints $x_{p,l} \geq x_{p,l+1}$ ($l \in [u_p - 1]$), we obtain an injective correspondence between x_p and $(x_{p,1}, \ldots, x_{p,u_i})$: $x_p = \sum_{l=1}^{u_p} x_{p,l}$. To model these constraints, we add the clause $(x_{p,l} \vee \neg x_{p,l+1})$ to $\mathcal{F}$, for each $l \in [u_p - 1]$.

Let $a_i x_p + b_i x_q \geq c_i$ be a constraint. We only state the case where $a_p, b_q > 0$ (for more details, see [21]). For $i \in [m]$ and $l \in \{0, \ldots, u_p\}$, let $\alpha_{i,l} = \lceil (c_i - l a_i)/b_i \rceil - 1$. The constraint can be expressed by adding the clauses to $\mathcal{F}$ as follows.

- $\big(x_{p,l+1} \vee x_{q,\alpha_{k,l+1}}\big)$, for every $l \in \{0, \ldots u_p - 1\}$ with $0 \leq \alpha_{i,l} < u_q$.
- $x_{p,l+1}$, for every $l \in \{0, \ldots, u_p - 1\}$ with $\alpha_{k,l} \geq u_q$.
- $x_{q,\alpha_{i,l}}$ for $l = u_p$ with $\alpha_{k,u_p} \geq 0$.

Next, we state how weights (and the function to be minimized) are encoded. Note that the weights appearing in the objective function are nonnegative integers. Let x_p be a variable with $w_p > 1$. To express the effect of setting x_p to 1 on the objective function, we introduce $w_p - 1$ additional variables $y_{p,1}, \ldots, y_{p,w_i - 1}$ and the clauses $(\neg x_p \vee y_{p,j})$ to $\mathcal{F}$, for all $j \in [w_i - 1]$.

Producing the clauses as a stream. Under the reasonable assumption that the clauses of $\mathcal{P}$ can each be stored in working memory, i.e. in $\mathrm{O}(f(k) \log n)$ bits of space, and by the construction of $\mathcal{F}$, it is easy to see that as a constraint of $\mathcal{P}$ arrives, we can construct the of corresponding clauses for that constraint in space bounded by $\mathrm{O}(g(k) \log n)$. The above discussions together with the algorithms of Section 2 and B.1, implies the proof of Theorem 2.11.

Fast Exact Algorithms Using Hadamard Product of Polynomials

V. Arvind
Institute of Mathematical Sciences (HBNI), Chennai, India
arvind@imsc.res.in

Abhranil Chatterjee
Institute of Mathematical Sciences (HBNI), Chennai, India
abhranilc@imsc.res.in

Rajit Datta
Chennai Mathematical Institute, Chennai, India
rajit@cmi.ac.in

Partha Mukhopadhyay
Chennai Mathematical Institute, Chennai, India
partham@cmi.ac.in

Abstract

Let C be an arithmetic circuit of poly(n) size given as input that computes a polynomial $f \in \mathbb{F}[X]$, where $X = \{x_1, x_2, \ldots, x_n\}$ and $\mathbb{F}$ is any field where the field arithmetic can be performed efficiently. We obtain new algorithms for the following two problems first studied by Koutis and Williams [13, 22, 14].

(*k*,*n*)-MLC: Compute the sum of the coefficients of all degree-k multilinear monomials in the polynomial f.

k-MMD: Test if there is a nonzero degree-k multilinear monomial in the polynomial f.

Our algorithms are based on the fact that the Hadamard product $f \circ S_{n,k}$, is the degree-k multilinear part of f, where $S_{n,k}$ is the k^{th} elementary symmetric polynomial.

- For (*k*,*n*)-MLC problem, we give a deterministic algorithm of run time $O^*(n^{k/2+c\log k})$ (where c is a constant), answering an open question of Koutis and Williams [14, ICALP'09]. As corollaries, we show $O^*(\binom{n}{\downarrow k/2})$-time exact counting algorithms for several combinatorial problems: k-Tree, t-Dominating Set, m-Dimensional k-Matching.
- For k-MMD problem, we give a randomized algorithm of run time $4.32^k \cdot \text{poly}(n,k)$. Our algorithm uses only poly(n,k) space. This matches the run time of a recent algorithm [8] for k-MMD which requires exponential (in k) space.

Other results include fast deterministic algorithms for (*k*,*n*)-MLC and k-MMD problems for depth three circuits.

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

Keywords and phrases Hadamard Product, Multilinear Monomial Detection and Counting, Rectangular Permanent, Symmetric Polynomial

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.9

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.04496`.

Acknowledgements We thank anonymous reviewers for their comments on an earlier version of this paper. We are particularly grateful to an anonymous reviewer for pointing out the combinatorial applications of Theorem 1 in exact counting.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 9; pp. 9:1–9:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Koutis and Williams [13, 22, 14] introduced and studied two algorithmic problems on arithmetic circuits. Given as input an arithmetic circuit C of $\mathrm{poly}(n)$ size computing a polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, the (k,n)-MLC problem is to compute the sum of the coefficients of all degree-k multilinear monomials in the polynomial f, and the k-MMD problem is to test if f has a nonzero degree-k multilinear monomial.

These problems are natural generalizations of the well-studied k-path detection and counting problems in a given graph [13] as well as several other combinatorial problems like k-Tree, t-Dominating Set, m-Dimensional k-Matching [14], well-studied in the parameterized complexity, reduce to these problems. In fact, the first randomized FPT algorithms for the decision version of these combinatorial problems were obtained from an $O^*(2^k)$ [1] algorithm for k-MMD for monotone circuits using group algebras [13, 22, 14]. Recently, Brand et al. [8] have given the first randomized FPT algorithm for k-MMD for general circuits that runs in time $O^*(4.32^k)$. Their method is based on exterior algebra and color coding [1].

In general, the exact counting versions of these problems are #W[1]-hard. For these counting problems, improvements to the trivial $O^*(n^k)$ time exhaustive search algorithm are known only in some cases (like counting k-paths) [6]. Since an improvement for (k,n)-MLC over exhaustive search will yield faster exact counting algorithms for all these problems, Koutis and Williams [14] pose this as an interesting open problem. They give an algorithm of run time $O^*(n^{k/2})$ to compute the *parity* of the sum of coefficients of degree-k multilinear monomials.

The techniques based on group algebra [13, 14] and exterior algebra [8] can be broadly classified as *multilinear algebra* techniques. We give a new approach to the k-MMD, (k,n)-MLC problems, and related problems. Our algorithm is based on computing the *Hadamard product of polynomials*. The Hadamard product (also known as Schur product) generally refers to Hadamard product of matrices and is used in matrix analysis. We consider the Hadamard product of polynomials (e.g., see [3]). Given polynomials $f, g \in \mathbb{F}[X]$, their Hadamard product is defined as $f \circ g = \sum_m ([m]f \cdot [m]g)m$, where $[m]f$ denotes the coefficient of monomial m in f.

The Hadamard product is a useful tool in *noncommutative* computation [3, 5]. A contribution of the present paper is to develop an efficient way to implement Hadamard product in the *commutative* setting which is useful for designing FPT and exact algorithms. As mentioned above, the Hadamard product has been useful in arithmetic circuit complexity results, e.g., showing hardness of the noncommutative determinant [5]. Transferring techniques from circuit complexity to algorithm design is an exciting area of research. We refer the reader to the survey article of Williams [21], see also [23].

This paper. We apply the Hadamard product of polynomials in the setting of commutative computation. This is achieved by combining earlier ideas [3, 5] with a symmetrization trick shown in Section 2. We then use it to design efficient algorithms for (k,n)-MLC, k-MMD and related problems.

Consider the elementary symmetric polynomial $S_{n,k}$ of degree k over the n variables $x_1, x_2, \ldots, x_n$. By definition, $S_{n,k}$ is the sum of all the degree-k multilinear monomials. Computing the Hadamard product of $S_{n,k}$ and a polynomial f *sieves out* precisely the degree-k multilinear part of f. This connection with the symmetric polynomial gives the following result.

[1] The O^* notation suppresses polynomial factors.

▶ **Theorem 1.** *The* (k,n)-MLC *problem for any arithmetic circuit C of* $\mathrm{poly}(n)$ *size, has a deterministic* $O^*(n^{k/2+c\log k})$ *time algorithm where c is a constant.*

The field $\mathbb{F}$ could be any field where the field operations can be efficiently computable. The above run time $O^*(n^{k/2+c\log k})$ (where c is a constant) beats the naive $O^*(n^k)$ bound, answering the question asked by Koutis and Williams [14].

An ingredient of the proof is a result in [5] that allows us to efficiently compute the Hadamard product of a noncommutative algebraic branching program (ABP) with a noncommutative polynomial f, even with only black-box access to f that allows evaluating f on matrix-valued inputs. The other ingredient is an algorithm of Björklund et al. [7] for evaluating rectangular permanent over noncommutative rings, that can be viewed as an algorithm for evaluating $S^*_{n,k}$ (a symmetrized noncommutative version of $S_{n,k}$) over matrices. Now, applying the routine conversion of a commutative circuit to an ABP, which incurs only a quasi-polynomial blow-up, we get a faster algorithm for (k,n)-MLC of general circuits. As applications of Theorem 1 we obtain improved counting algorithms for k-Tree, t-Dominating Set, and m-Dimensional k-Matching.

The next algorithmic result we obtain is the following.

▶ **Theorem 2.** *The* k-MMD *problem for any arithmetic circuit C of* $\mathrm{poly}(n)$ *size, has a randomized* $O^*(4.32^k)$ *time and polynomial space-bounded algorithm.*

Again, the field $\mathbb{F}$ could be any field where the field operation can be efficiently computable. We briefly sketch the proof idea. Suppose that C is the input arithmetic circuit computing a homogeneous polynomial f of degree k. We essentially show that k-MMD is reducible to checking if the Hadamard product $f \circ C'$ is nonzero for some circuit C' from a collection of homogeneous degree-k depth two circuits. This collection of depth two circuits arises from the application of color coding [1]. Furthermore, the commutative Hadamard product $f \circ C'$ turns out to be computable in $O^*(2^k)$ time by a symmetrization trick combined with Ryser's formula for the permanent. The overall running time (because of trying several choices for C') turns out to be $O^*(4.32^k)$. Finally, checking if $f \circ C'$ is nonzero reduces to an instance of polynomial identity testing which can be solved in randomized polynomial time using Demillo-Lipton-Schwartz-Zippel Lemma [9, 24, 18]. The technique based on Hadamard product seems to be quite different than the exterior algebra based technique. Another difference is that, our algorithm uses $\mathrm{poly}(n,k)$ space whereas the algorithm in [8] takes exponential space.

Next, we state the results showing fast deterministic algorithms for depth-three circuits. We use the notation $\Sigma^{[s]}\Pi^{[k]}\Sigma$ to denote depth three circuits of top Σ gate fan-in s and the Π gates compute the product of k homogeneous linear forms over X.

▶ **Theorem 3.** *Given any homogeneous depth three $\Sigma^{[s]}\Pi^{[k]}\Sigma$ circuit of degree k, the* (k,n)-MLC *problem can be solved in deterministic $O^*(2^k)$ time. Over $\mathbb{Z}$, the* k-MMD *problem can be solved in deterministic $O^*(4^k)$ time. Over finite fields,* k-MMD *problem can be solved in deterministic* $e^k k^{O(\log k)}(2^{ck} + 2^k) \cdot \mathrm{poly}(n,k,s)$ *time, where* $c \leq 5$.

Here the key observation is that we can efficiently compute the commutative Hadamard product of a depth three circuit with *any* circuit. It is well-known that the elementary symmetric polynomial $S_{n,k}$ can be computed using an algebraic branching program of size $\mathrm{poly}(n,k)$.

We compute the Hadamard product of the given depth three circuit with that homogeneous branching program for $S_{n,k}$, and check whether the resulting depth three circuit is identically zero or not. The same idea yields the algorithm to compute the sum of the coefficients of the multilinear terms as well.

Related Work. Soon after the first version of our paper [2] appeared in ArXiv, an independent work [16, v1] [2] also considers the k-MMD and (k,n)-MLC problems. The main ingredient of [16] is the application of a nontrivial Waring decomposition over rationals of symmetric polynomials [15] which does not have any known analogue for small finite fields. The algorithms obtained for k-MMD and (k,n)-MLC are faster ($O^*(4.08^k)$ time for k-MMD and $O^*(n^{k/2})$ for (k,n)-MLC). In comparison, our algorithms also work for all finite fields. As already mentioned, the algorithm of Koutis and Williams [14] for (k,n)-MLC works over $\mathbb{F}_2$ and the run time is $O^*(n^{k/2})$. In this sense, our algorithm for (k,n)-MLC can also be viewed as a generalization that does not depend on the characteristic of the ground field. It is to be noted that, over fields of small characteristic a Waring decomposition of the input polynomial may not be available. For example, over $\mathbb{F}_2$ the polynomial xy has no Waring decomposition.

Organization. The paper is organized as follows. In Section 2 we explain the Hadamard product framework. The proof of Theorem 1 and its consequences are given in Section 3. Section 4 contains the the proof of Theorem 2. The proof of Theorem 3 can be found in the full version in ArXiv.

2 Hadamard Product Framework

Computing the Hadamard product of two commutative polynomials is, in general, computationally hard. This can be observed from the fact that the Hadamard product of the determinant polynomial with itself is the permanent polynomial. Nevertheless, we develop a method for some special cases, that is efficient with degree k as the fixed parameter, for computing the *scaled* Hadamard product of commutative polynomials.

▶ **Definition 4.** *The* scaled *Hadamard product of polynomials $f, g \in \mathbb{F}[X]$ is defined as*

$$f \circ^s g = \sum_m (m! \cdot [m]f \cdot [m]g)\ m,$$

where for monomial $m = x_{i_1}^{e_1} x_{i_2}^{e_2} \dots x_{i_r}^{e_r}$ we define $m! = e_1! \cdot e_2! \cdots e_r!$.

Computing the scaled Hadamard product is key to our algorithmic results for k-MMD and (k,n)-MLC. Broadly, it works as follows: we transform polynomials f and g to suitable *noncommutative* polynomials. We compute their (noncommutative) Hadamard product efficiently [3, 5], and we finally recover the scaled commutative Hadamard product $f \circ^s g$ (or evaluate it at a desired point $\vec{a} \in \mathbb{F}^n$).

Suppose $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ is a homogeneous degree-k polynomial given by a circuit C. We can define its noncommutative version C^{nc} which computes the noncommutative homogeneous degree-k polynomial $\hat{f} \in \mathbb{F}\langle y_1, y_2, \dots, y_n \rangle$ as follows.

▶ **Definition 5.** *Given a commutative circuit C computing a polynomial in $\mathbb{F}[x_1, x_2, ..., x_n]$, the noncommutative version of C, C^{nc} is the noncommutative circuit obtained from C by fixing an ordering of the inputs to each product gate in C and replacing x_i by the noncommuting variable $y_i : 1 \leq i \leq n$.*

[2] See the final version [16] to be appeared in FOCS 2019.

Let X_k denote the set of all degree-k monomials over X. As usual, Y^k denotes all degree-k noncommutative monomials (i.e., words) over Y. Each monomial $m \in X_k$ can appear as different noncommutative monomials $\hat{m}$ in $\hat{f}$. We use the notation $\hat{m} \to m$ to denote that $\hat{m} \in Y^k$ will be transformed to $m \in X_k$ by substituting x_i for $y_i, 1 \leq i \leq n$. Then, we observe the following, $[m]f = \sum_{\hat{m}\to m}[\hat{m}]\hat{f}$.

The noncommutative circuit C^{nc} is not directly useful for computing Hadamard product. However, the following symmetrization helps. We first explain how permutations $\sigma \in S_k$ act on the set Y^k of degree-k monomials (and hence, by linearity, act on homogeneous degree k polynomials).

For each monomial $\hat{m} = y_{i_1}y_{i_2}\cdots y_{i_k}$, the permutation $\sigma \in S_k$ maps $\hat{m}$ to the monomial $\hat{m}^\sigma$ defined as $\hat{m}^\sigma = y_{i_{\sigma(1)}}y_{i_{\sigma(2)}}\cdots y_{i_{\sigma(k)}}$. By linearity, $\hat{f} = \sum_{\hat{m}\in Y^k}[\hat{m}]\hat{f}\cdot\hat{m}$ is mapped by σ to the polynomial, $\hat{f}^\sigma = \sum_{\hat{m}\in Y^k}[\hat{m}]\hat{f}\cdot\hat{m}^\sigma$.

The *symmetrized polynomial* of f, f^*, is degree-k homogeneous polynomial $f^* = \sum_{\sigma\in S_k}\hat{f}^\sigma$. We now explain the use of symmetrization in computing the scaled Hadamard product $f \circ^s g$.

▶ **Proposition 6.** *For a homogeneous degree-k commutative polynomial $f \in \mathbb{F}[X]$ given by circuit C, and its noncommutative version C^{nc} computing polynomial $\hat{f} \in \mathbb{F}\langle Y\rangle$, consider the* symmetrized *noncommutative polynomial $f^* = \sum_{\sigma\in S_k}\hat{f}^\sigma$. Then for each monomial $m \in X_k$ and each word $m' \in Y^k$ such that $m' \to m$, we have: $[m']f^* = m!\cdot[m]f$.*

Proof. Let $f = \sum_m[m]f\cdot m$ and $\hat{f} = \sum_{\hat{m}}[\hat{m}]\hat{f}\cdot\hat{m}$. Notice that $[m]f = \sum_{\hat{m}\to m}[\hat{m}]\hat{f}$. Now, we write $f^* = \sum_{m'}[m']f^*\cdot m'$. The group S_k acts on Y^k (degree k words in Y) by permuting the positions. Suppose $m = x_{i_1}^{e_1}\cdots x_{i_q}^{e_q}$ is a type $\boldsymbol{e} = (e_1,\ldots,e_q)$ degree k monomial over X and $m' \to m$. Then, by the *Orbit-Stabilizer lemma* the orbit $O_{m'}$ of m' has size $\frac{k!}{m!}$. It follows that

$$[m']f^* = \sum_{\hat{m}\in O_{m'}} m!\cdot[\hat{m}]\hat{f} = m!\sum_{\hat{m}\to m}[\hat{m}]\hat{f} = m!\cdot[m]f.$$

It is important to note that for some $\hat{m} \in Y^k$ such that $\hat{m} \to m$, even if $[\hat{m}]\hat{f} = 0$ then also $[\hat{m}]f^* = m!\cdot[m]f$. ◀

Next, we show how to use Proposition 6 to compute scaled Hadamard product in the commutative setting via noncommutative Hadamard product. We note that given a commutative circuit C computing f, the noncommutative polynomial $\hat{f}$ depends on the circuit structure of C. However, f^* depends only on the polynomial f.

▶ **Lemma 7.** *Let C be a circuit for a homogeneous degree-k polynomial $g \in \mathbb{F}[X]$. For any homogeneous degree-k polynomial $f \in \mathbb{F}[X]$, to compute a circuit for $f \circ^s g$ efficiently, it suffices to compute a circuit for $f^* \circ \hat{g}$ efficiently where $\hat{g}$ is the polynomial computed by the noncommutative circuit C^{nc}. Moreover, given any point $\vec{a} \in \mathbb{F}^n$, $(f \circ^s g)(\vec{a}) = (f^* \circ \hat{g})(\vec{a})$.*

Proof. We write $f = \sum_m[m]f\cdot m$ and $g = \sum_{m'}[m']g\cdot m'$, and notice that $f \circ^s g = \sum_m m!\cdot[m]f\cdot[m]g\cdot m$.

Suppose the polynomial computed by C^{nc} is $\hat{g}(Y) = \sum_{m\in X_k}\sum_{\hat{m}\to m}[\hat{m}]\hat{g}\cdot\hat{m}$. By Proposition 6, the noncommutative polynomial $f^*(Y) = \sum_{m\in X_k}\sum_{\hat{m}\to m} m!\cdot[m]f\cdot\hat{m}$. Hence,

$$(f^*\circ\hat{g})(Y) = \sum_{m\in X_k}\sum_{\hat{m}\to m} m!\cdot[m]f\cdot[\hat{m}]g\cdot\hat{m} = \sum_{m\in X_k} m!\cdot[m]f\sum_{\hat{m}\to m}[\hat{m}]g\cdot\hat{m}.$$

Therefore, using any commutative substitution (i.e. by substituting the Y variables by X variables), we get back a commutative circuit for $f \circ^s g$. Moreover, given a point $\vec{a} \in \mathbb{F}^n$,

$$(f^* \circ \hat{g})(\vec{a}) = \sum_{m \in X_k} m! \cdot [m]f \sum_{\hat{m} \to m} [\hat{m}]g \cdot \hat{m}(\vec{a}) = \sum_{m \in X_k} m! \cdot [m]f \cdot m(\vec{a}) \sum_{\hat{m} \to m} [\hat{m}]g.$$

From the definition, $[m]g = \sum_{\hat{m} \to m} \hat{m}[\hat{g}]$. Hence, $(f^* \circ \hat{g})(\vec{a}) = \sum_{m \in X_k} m! \cdot [m]f \cdot m(\vec{a})[m]g = (f \circ^s g)(\vec{a})$. ◀

3 The Sum of Coefficients of Multilinear Monomials

In this section we prove Theorem 1. As already sketched in Section 1, the main conceptual step is to apply the symmetrization trick to reduce the (k,n)-MLC problem to evaluating rectangular permanent over a suitable matrix ring. Then we use a result of [7] to solve the instance of rectangular permanent evaluation problem. As corollaries of our technique, we improve the running time of exact counting of several combinatorial problems studied in [14].

Before we prove the theorem, let us recall the definition of an ABP. An *algebraic branching program* (ABP) is a directed acyclic graph with one in-degree-0 vertex called *source*, and one out-degree-0 vertex called *sink*. The vertex set of the graph is partitioned into layers $0, 1, \ldots, \ell$, with directed edges only between adjacent layers (i to $i+1$). The source and the sink are at layers zero and ℓ respectively. Each edge is labeled by a linear form over variables $x_1, x_2, \ldots, x_n$. The polynomial computed by the ABP is the sum over all source-to-sink directed paths of the product of linear forms that label the edges of the path. An ABP is *homogeneous* if all edge labels are homogeneous linear forms. ABPs can be defined in both commutative and noncommutative settings. Equivalently, a homogeneous ABP of width w computing a degree-k polynomial over X can be thought of as the $(1, w)^{th}$ entry of the product of $w \times w$ matrices $M_1 \cdots M_k$ where entries of each M_i are homogeneous linear forms over X. By $[x_j]M_i$, we denote the $w \times w$ matrix over $\mathbb{F}$, such that $(p,q)^{th}$ entry of the matrix,$([x_j]M_i)(p,q) = [x_j](M_i(p,q))$, the coefficient of x_j in the linear form of the $(p,q)^{th}$ entry of M_i.

We now define the permanent of a rectangular matrix. The permanent of a rectangular $k \times n$ matrix $A = (a_{ij})$, with $k \leq n$ is defined as $\mathrm{rPer}(A) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^{k} a_{i,\sigma(i)}$ where $I_{k,n}$ is the set of all injections from $[k]$ to $[n]$. Also, we define the noncommutative polynomial $S^*_{n,k}$ as $S^*_{n,k}(y_1, y_2, \ldots, y_n) = \sum_{T \subseteq [n], |T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} y_{\sigma(i)}$ which is the symmetrized version of the elementary symmetric polynomial $S_{n,k}$ as defined in Proposition 6. Given a set of matrices $M_1, \ldots, M_n$ define the rectangular matrix $A = (a_{i,j})_{i \in [k], j \in [n]}$ such that $a_{i,j} = M_j$. Now we make the following crucial observation.

▶ **Observation 8.**

$$S^*_{n,k}(M_1, \ldots, M_n) = \mathrm{rPer}(A).$$

We use a result from [7], that shows that over *any* ring R, the permanent of a rectangular $k \times n$ matrix can be evaluated using $O^*(k\binom{n}{\downarrow k/2})$ ring operations. In particular, if R is a matrix ring $M_s(\mathbb{F})$, the algorithm runs in time $O(k\binom{n}{\downarrow k/2}\,\mathrm{poly}(n,s))$. Now we are ready to prove Theorem 1.

Proof. Let us first proof a special case of the theorem when the polynomial f is given by an ABP B of width s. Notice that, we can compute the sum of the coefficients of the degree-k multilinear terms by evaluating $(f \circ S_{n,k})(\vec{1})$. Now to compute the Hadamard product efficiently, we transfer the problem to the noncommutative domain. Let B^{nc} defines the noncommutative version of the commutative ABP B computing the polynomial f. From Lemma 7, it suffices to compute $(B^{nc} \circ S^*_{n,k})(\vec{1})$. Now, the following lemma reduces this to evaluating $S^*_{n,k}$ over matrix ring. We recall the following result from [5].

▶ **Lemma 9** (Theorem 2 of [4]). *Let f be a homogeneous degree-k noncommutative polynomial in $\mathbb{F}\langle Y\rangle$ and B be an ABP of width w computing a homogeneous degree-k polynomial $g = (M_1 \cdots M_k)(1, w)$ in $\mathbb{F}\langle Y\rangle$. Then $(f \circ g)(\vec{1}) = (f(A^B_1, \ldots, A^B_n))(1, (k+1)w)$ where for each $i \in [n]$, A^B_i is the following $(k+1)w \times (k+1)w$ block superdiagonal matrix,*

$$A^B_i = \begin{bmatrix} 0 & [y_i]M_1 & 0 & \ldots & 0 \\ 0 & 0 & [y_i]M_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & [y_i]M_k \\ 0 & 0 & 0 & \ldots & 0 \end{bmatrix}.$$

To see the proof, for any monomial $m = y_{i_1} y_{i_2} \cdots y_{i_k} \in Y^k$,

$$(A^B_{i_1} A^B_{i_2} \cdots A^B_{i_k})(1, (k+1)w) = ([y_{i_1}]M_1 \cdot [y_{i_2}]M_2 \cdots [y_{i_k}]M_k)(1, w) = [m]g,$$

from the definition. Hence, we have,

$$\begin{aligned} f(A^B_1, A^B_2, \ldots, A^B_n)(1, (k+1)w) &= \sum_{m \in Y^k} [m]f \cdot m(A^B_{i_1}, A^B_{i_2}, \ldots, A^B_{i_k})(1, (k+1)w) \\ &= \sum_{m \in Y^k} [m]f \cdot [m]g. \end{aligned}$$

Now, we construct a $k \times n$ rectangular matrix $A = (a_{i,j})_{i \in [k], j \in [n]}$ from the given ABP B^{nc} setting $a_{i,j} = A^{B^{nc}}_j$ as defined. Using Observation 8, we now have,

$$\mathrm{rPer}(A)(1, (k+1)s) = S^*_{n,k}(A^{B^{nc}}_1, \ldots, A^{B^{nc}}_n)(1, (k+1)s) = (S^*_{n,k} \circ B^{nc})(\vec{1}) = (S_{n,k} \circ^s B)(\vec{1}).$$

Hence combining the algorithm of Björklund et al. for evaluating rectangular permanent over noncommutative ring [7] with Lemma 9, we can evaluate the sum of the coefficients deterministically in time $O(k\binom{n}{\downarrow k/2} \operatorname{poly}(s, n))$.

Now, we are ready to prove the general case. It uses the following standard transformation from circuit to ABP [20, 19] and reduces the problem to the ABP case again. Given an arithmetic circuit of size s' computing a polynomial f of degree k, f can also be computed by a homogeneous ABP of size $s'^{O(\log k)}$. Hence given a polynomial f by a poly(n) sized circuit, we first get a circuit of poly(n) size for degree-k part of f using standard method of homogenization [19]. Then we convert the circuit to a homogeneous ABP of size $n^{O(\log k)}$. Then, we apply the first part of the proof on the newly constructed ABP. Notice that the entire computation can be done in deterministic $O^*(n^{k/2 + c\log k})$ for some constant c. ◀

Some Applications

As immediate consequence of Theorem 1, we improve the counting complexity of several combinatorial problems studied in [14]. To the best of our knowledge, nothing better than the trivial exhaustive search algorithm were known for the counting version of these problems. We start with the k-Tree problem.

▶ **Corollary 10.** *Given a tree T of k nodes and a graph G of n nodes, we can count the number of (not necessarily induced) copies of T in G in deterministic $O^*(\binom{n}{\downarrow k/2})$ time.*

Proof. Let us define $Q = \sum_{j \in V(T), i \in V(G)} C_{T,i,j}$, following [14] where if $|V(T)| = 1$, we define $C_{T,i,j} = x_j$ and if $|V(T)| > 1$, let $T_{i,1}, \ldots, T_{i,\ell}$ be the connected subtrees of T remaining after node$i$ is removed from T. For each $t \in [\ell]$, let $n_{i,t} \in [k]$ be the (unique) node in $T_{i,t}$ that is a neighbour of i in T, then we define

$$C_{T,i,j} = \prod_{t=1}^{\ell} \left(\sum_{j' : (j,j') \in E(G)} x_j \cdot C_{T_{i,t}, n_{i,t}, j'} \right).$$

By the result of [14], it is known that to solve the k-Tree problem it is sufficient to count the number of multilinear terms in Q. Following Theorem 1, it suffices to show that Q has a $\text{poly}(n,k)$ sized ABP. It is enough to show that $C_{T,i,j}$ has a $\text{poly}(n,k)$ sized ABP and the ABP for Q follows easily. We construct an ABP for each $C(T,i,j)$ of size $\text{poly}(n,k)$ by induction on size of T. Suppose $C_{T,i,j}$ has such small ABP for $|V(T)| \leq p$. Then, for $V(T) = p+1$, it is clear from the definition that $C(T,i,j)$ will also have a small ABP. Therefore, the polynomial Q will also have an ABP of size $\text{poly}(n,k)$. ◀

The second application is for t-Dominating Set problem.

▶ **Corollary 11.** *Given a graph $G = (V,E)$, we can count the number of sets S of size k that dominates at least t nodes in G in $O^*(\binom{n}{\downarrow t/2})$ deterministic time.*

Proof. Following [14], define

$$P(X,z) = \left(\sum_{i \in V} \left((1 + zx_i) \prod_{j:(i,j) \in E} (1 + zx_j) \right) \right)^k.$$

We inspect $[z^t]P(X,z)$ which is a homogeneous degree t polynomial over X, call it $Q(X)$. As $P(X,z)$ has a small ABP of $\text{poly}(n,k)$ size substituing z by any scalar, we obtain an ABP of size $\text{poly}(n,k)$ for $Q(X)$ also by interpolation. Then, we use the standard method to homogenize the ABP and apply Theorem 1 to count the number of multilinear terms. This is sufficient to solve the problem by the result of [14]. ◀

The final application is regarding m-Dimensional k-Matching problem.

▶ **Corollary 12.** *Given mutually disjoint sets U_i, $i \in [m]$, and a collection C of m-tuples from $U_1 \times \cdots \times U_m$, we can count the number of sub-collection of k mutually disjoint m-tuples in C in deterministic $O^*(\binom{n}{\downarrow (m-1)k/2})$ time.*

Proof. Following [14] , encode each element u in $U = \cup_{i=2}^{m} U_i$ by a variable $x_u \in X$. Encode each m-tuple $t = (u_1, \ldots, u_m) \in C \subseteq U_1 \times \cdots \times U_m$ by the monomial $M_t = \prod_{i=2}^{m} x_{u_i}$. Assume $U_1 = \{u_{1,1}, \ldots, u_{1,n}\}$, and let $T_j \subseteq C$ denote the subset of m-tuples whose first coordinate is $u_{1,j}$. Consider the polynomial

$$P(X,z) = \prod_{j=1}^{n} \left(1 + \sum_{t \in T_j} (z \cdot M_t) \right).$$

Clearly, $P(X,z)$ has an ABP of size $\text{poly}(n,m)$. Let $Q(X) = [z^k]P(X,z)$, we can obtain a small ABP of size $\text{poly}(n,m,k)$ for $Q(X)$ by interpolation. Now, we homogenize the ABP and apply Theorem 1 to count the number of multilinear terms which is sufficient by the result of [14]. ◀

Hardness for Computing Rectangular Permanent over *any* Ring

In [7], it is shown that a $k \times n$ rectangular permanent can be evaluated over commutative rings and commutative semirings in $O(h(k) \cdot \text{poly}(n, k))$ time for some computable function h . In other words, the problem is in FPT parameterized by the number of rows. An interesting question is to ask whether one can get any FPT algorithm when the entries are from noncommutative rings (in particular, matrix rings). We prove that such an algorithm is unlikely to exist. We show that counting the number of k-paths in a graph G, a well-known #W[1]-complete problem, reduces to this problem. So, unless ETH fails we do not have such an algorithm [10].

▶ **Theorem 13.** *Given a $k \times n$ matrix X with entries $X_{ij} \in \mathbb{M}_{t\times t}(\mathbb{Q})$, computing the rectangular permanent of X is #W[1]-hard with k as the parameter where $t = (k+1)n$ under polynomial time many-one reductions.*

Proof. If we have an algorithm to compute the permanent of a $k \times n$ matrix over noncommutative rings which is FPT in parameter k, that yields an algorithm which is FPT in k for evaluating the polynomial $S^*_{n,k}$ on matrix inputs. This follows from Observation 8. Now, given a graph G we can compute a homogeneous ABP of width n and k layers for the graph polynomial C_G defined as follows. Let $G(V, E)$ be a directed graph with n vertices where $V(G) = \{v_1, v_2, \ldots, v_n\}$. A k-walk is a sequence of k vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ where $(v_{i_j}, v_{i_{j+1}}) \in E$ for each $1 \le j \le k-1$. A k-path is a k-walk where no vertex is repeated. Let A be the adjacency matrix of G, and let $y_1, y_2, \ldots, y_n$ be noncommuting variables. Define an $n \times n$ matrix B

$$B[i,j] = A[i,j] \cdot y_i, \quad 1 \le i, j \le n.$$

Let $\vec{1}$ denote the all 1's vector of length n. Let $\vec{y}$ be the length n vector defined by $\vec{y}[i] = y_i$. The *graph polynomial* $C_G \in \mathbb{F}\langle Y\rangle$ is defined as

$$C_G(Y) = \vec{1}^T \cdot B^{k-1} \cdot \vec{y}.$$

Let W be the set of all k-walks in G. The following observation is folklore.

▶ **Observation 14.**

$$C_G(Y) = \sum_{v_{i_1} v_{i_2} \cdots v_{i_k} \in W} y_{i_1} y_{i_2} \cdots y_{i_k}.$$

Hence, G contains a k-path if and only if the graph polynomial C_G contains a multilinear term.

Clearly the number of k-paths in G is equal to $(C_G \circ S_{n,k})(\vec{1})$. By Lemma 7, we know that it suffices to compute $({C_G}^{nc} \circ S^*_{n,k})(\vec{1})$. We construct $kn \times kn$ matrices $A_1, \ldots, A_n$ from the ABP of ${C_G}^{nc}$ following Lemma 9. Then from Lemma 9, we know that $({C_G}^{nc} \circ S^*_{n,k})(\vec{1}) = S^*_{n,k}(A_1, \ldots, A_n)(1, t)$ where $t = (k+1)n$. So if we have an algorithm which is FPT in k for evaluating $S^*_{n,k}$ over matrix inputs, we also get an algorithm to count the number of k-paths in G in FPT(k) time. ◀

4 Multilinear Monomial Detection

In this section, we prove Theorem 2. Apart from being a new technique, the Hadamard product based algorithm runs in polynomial space and does not depend on the characteristic of the ground field. This is in contrast with the exterior algebra based approach [8] and Waring rank based approach [16].

We first recall that the Hadamard product of a noncommutative circuit and a noncommutative ABP can be computed efficiently. The proof is similar to the proof of Lemma 9.

▶ **Lemma 15** (Corollary 4 of [5]). *Given a homogeneous noncommutative circuit of size S' for $f \in \mathbb{F}\langle y_1, y_2, \ldots, y_n\rangle$ and a homogeneous noncommutative ABP of size S for $g \in \mathbb{F}\langle y_1, y_2, \ldots, y_n\rangle$, we can compute a noncommutative circuit of size $O(S^3S')$ for $f \circ g$ in deterministic $S^3S' \cdot \mathrm{poly}(n,k)$ time where k is the degree upper bound for f and g.*

Now we give an algorithm for computing the Hadamard product for a special case in the commutative setting. Any depth two $\Pi^{[k]}\Sigma$ circuit computes the product of k homogeneous linear forms over the input set of variables X.

▶ **Lemma 16.** *Given an arithmetic circuit C of size s computing $g \in \mathbb{F}[X]$, and a homogeneous $\Pi^{[k]}\Sigma$ circuit computing $f \in \mathbb{F}[X]$, and any point $\vec{a} \in \mathbb{F}^n$, we can evaluate $(f \circ^s g)(\vec{a})$ in $O^*(2^k)$ time and in polynomial space.*

Proof. By standard homogenization technique [19] we can extract the homogeneous degree-k component of C and thus we can assume that C computes a homogeneous degree-k polynomial. Write $f = \prod_{j=1}^{k} L_j$, for homogeneous linear forms L_j. The corresponding noncommutative polynomial $\hat{f}$ is defined by the natural order of the j indices (and replacing x_i by y_i for each i).

▷ Claim 17. The noncommutative polynomial f^* has a (noncommutative) $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ circuit, which we can write as $f^* = \sum_{i=1}^{2^k} C_i$, where each C_i is a (noncommutative) $\Pi^{[k]}\Sigma$ circuit.

Before we prove the claim, we show that it easily yields the desired algorithm: First we notice that

$$C^{nc} \circ f^* = \sum_{i=1}^{2^k} C^{nc} \circ C_i.$$

Now, by Lemma 15, we can compute in $\mathrm{poly}(n,s,k)$ time a $\mathrm{poly}(n,s,k)$ size circuit for the (noncommutative) Hadamard product $C^{nc} \circ C_i$. As argued in the proof of Lemma 7, for any $\vec{a} \in \mathbb{F}^n$ we have

$$(g \circ^s f)(\vec{a}) = (C \circ^s f)(\vec{a}) = (C^{nc} \circ f^*)(\vec{a}).$$

Thus, we can evaluate $(g \circ^s f)(\vec{a})$ by incrementally computing $(C^{nc} \circ C_i)(\vec{a})$ and adding up for $1 \leq i \leq 2^k$. This can be clearly implemented using only polynomial space. ◀

Now, we prove the above claim. Consider $f = L_1 L_2 \cdots L_k$. Then $\hat{f} = \hat{L}_1 \hat{L}_2 \cdots \hat{L}_k$, where $\hat{L}_j$ is obtained from L_j by replacing variables x_i with the noncommutative variable y_i for each i. We will require the following observation.

▶ **Observation 18.**

$$f^* = \sum_{\sigma \in S_k} \hat{L}_{\sigma(1)} \hat{L}_{\sigma(2)} \cdots \hat{L}_{\sigma(k)}.$$

Proof. Let us prove the claim, monomial by monomial. Fix a monomial m' in f^* such that $m' \to m$. Suppose $m' = y_{i_1} y_{i_2} \ldots y_{i_k}$. Note that, $m = x_{i_1} x_{i_2} \ldots x_{i_k}$. Recall from Proposition 6, $[m']f^* = m! \cdot [m]f$. Now, the coefficient of m' in $\sum_{\sigma \in S_k} \prod_{j=1}^{k} \hat{L}_{\sigma(j)}$ is

$$[m'] \left(\sum_{\sigma \in S_k} \prod_{j=1}^{k} \hat{L}_{\sigma(j)} \right) = \sum_{\sigma \in S_k} \prod_{j=1}^{k} [y_{i_j}] \hat{L}_{\sigma(j)}.$$

Let us notice that, $[y_{i_j}]\hat{L}_{\sigma(j)} = [x_{i_j}]L_{\sigma(j)}$. Hence,

$$[m']\left(\sum_{\sigma\in S_k}\prod_{j=1}^{k}\hat{L}_{\sigma(j)}\right) = \sum_{\sigma\in S_k}\prod_{j=1}^{k}[x_{i_j}]L_{\sigma(j)}. \qquad \blacktriangleleft$$

Now we observe the following easy fact.

▶ **Observation 19.** *For a degree k monomial $m = x_{i_1}x_{i_2}\cdots x_{i_k}$ (where the variables can have repeated occurrences) and a homogeneous $\Pi^{[k]}\Sigma$ circuit $C = \prod_{j=1}^{k} L_j$, the coefficient of monomial m in C is given by $m!\cdot[m]C = \sum_{\sigma\in S_k}\prod_{j=1}^{k}([x_{i_j}]L_{\sigma(j)})$.*

Proof of Claim 17. Now, the claim directly follows from Observation 19 as $\sum_{\sigma\in S_k}\prod_{j=1}^{k}[x_{i_j}]L_{\sigma(j)} = m!\cdot[m]f$.

Now define the $k\times k$ matrix T such that each row of T_i is just the linear forms $\hat{L}_1,\hat{L}_2,\ldots,\hat{L}_k$ appearing in f. The (noncommutative) permanent of T is given by $\mathrm{Perm}(T) = \sum_{\sigma\in S_k}\prod_{j=1}^{k}\hat{L}_{\sigma(j)}$, which is just f^*.

We now apply Ryser's formula [17] (noting the fact that it holds for the noncommutative permanent too), to express $\mathrm{Perm}(T)$ as a depth-3 homogeneous noncommutative $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ formula. It follows that $f^* = \mathrm{Perm}(T)$ has a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ noncommutative formula. ◁

We include a proof of Observation 19 for completeness.

Proof. We assume, without loss of generality, that repeated variables are adjacent in the monomial $m = x_{i_1}x_{i_2}\cdots x_{i_k}$. More precisely, suppose the first e_1 variables are x_{j_1}, and the next e_2 variables are x_{j_2} and so on until the last e_q variables are x_{j_q}, where the q variables $x_{j_k}, 1\le k\le q$ are all distinct.

We notice that the monomial m can be generated in C by first fixing an order $\sigma : [k]\mapsto[k]$ for multiplying the k linear forms as $L_{\sigma(1)}L_{\sigma(2)}\cdots L_{\sigma(k)}$, and then multiplying the coefficients of variable $x_{i_j}, 1\le j\le k$ picked successively from linear forms $L_{\sigma(j)}, 1\le j\le k$. However, these $k!$ orderings repeatedly count terms.

We claim that each distinct product of coefficients term is counted exactly $m!$ times. Let $E_j\subseteq[k]$ denote the interval $E_j = \{\ell \mid e_{j-1}+1\le\ell\le e_j\}, 1\le j\le q$, where we set $e_0 = 0$.

Now, to see the claim we only need to note that two permutations $\sigma,\tau\in S_k$ give rise to the same product of coefficients term if and only if $\sigma(E_j) = \tau(E_j), 1\le j\le q$. Thus, the number of permutations τ that generate the same term as σ is $m!$.

Therefore the sum of product of coefficients $\sum_{\sigma\in S_k}\prod_{j=1}^{k}([x_{i_j}]L_{\sigma(j)})$ is same as $m!\cdot[m]C$, which completes the proof. ◀

▶ Remark 20. Over rationals, computing $f\circ^s g$, when g is a $\Pi^{[k]}\Sigma$ circuit, can also be done by computing g^* using Fischer's identity [11]. However, Lemma 16 also works over finite fields.

Now we are ready to prove Theorem 2.

Proof. By homogenization, we can assume that C computes a homogeneous degree k polynomial f.

We go over a collection of colorings $\{\zeta_i : [n]\to[k]\}$ chosen uniformly at random and define a $\Pi^{[k]}\Sigma$ formula $P_i = \prod_{j=1}^{k}\sum_{\ell:\zeta_i(\ell)=j} x_\ell$ for each colouring ζ_i. A monomial is *covered* by a coloring ζ_i if the monomial is nonzero in P_i. The probability that a random coloring covers a given degree-k multilinear monomial is $\frac{k!}{k^k}\approx e^{-k}$. Hence, for a collection of $O^*(e^k)$ many colorings $\{\zeta_i : [n]\to[k]\}$ chosen uniformly at random, with constant probability all the multilinear terms of degree k are covered.

For each coloring ζ_i, we construct a circuit $C'_i = C \circ^s P_i$.

Notice that we are interested only in multilinear monomials and for each such monomial m, the additional multiplicative factor $m! = 1$. Also, the coefficient of each monomial is exactly 1 in each P_i, and if f contains a multilinear term then it will be covered by *some* P_i. Now we perform PIT test on each C'_i using Demillo-Lipton-Schwartz-Zippel Lemma [9, 24, 18] in randomized polynomial time to complete the procedure. More precisely, we pick a random $\vec{a} \in \mathbb{F}^n$ and evaluate C'_i on that point. Notice that, by the proof of Lemma 16, it is easy to see that $C'_i(\vec{a})$ can be computed deterministically in time $2^k \cdot \mathrm{poly}(n, s)$ time and $\mathrm{poly}(n, k)$ space.[3]

To improve the run time from $O^*((2e)^k)$ to $O^*(4.32^k)$, we can use the idea of Hüffner et al. [12][4]. The key idea is that, using more than k colors we would reduce the number of colorings and hence the number of $\Pi\Sigma$ circuits, but it would increase the formal degree of each P_i. Following [12], we use $1.3k$ many colors and each P_i will be a $\Pi^{[1.3k]}\Sigma$ circuit. For each coloring $\zeta_i : [n] \to [1.3k]$ chosen uniformly at random, we define the following $\Pi^{[1.3k]}\Sigma$ circuit, $P_i(x_1, x_2, \ldots, x_n, z_1, \ldots, z_{1.3k}) = \prod_{j=1}^{1.3k} \left(\sum_{\ell:\zeta_i(\ell)=j} x_\ell + z_j\right)$.

Since each P_i is of degree $1.3k$, we need to modify the circuit C to another circuit C' of degree $1.3k$ in order to apply Hadamard products. The key idea is to define the circuits $C' \in \mathbb{F}[X, Z]$ as follows:

$$C'(X, Z) = C(X) \cdot S_{1.3k,0.3k}(z_1, \ldots, z_{1.3k})$$

where $S_{1.3k,0.3k}(z_1, \ldots, z_{1.3k})$ is the elementary symmetric polynomial of degree $0.3k$ over the variables $z_1, \ldots, z_{1.3k}$. By the result of [12], for $O^*(1.752^k)$ many random colorings with high probability each multilinear monomial in C will be covered by the monomials of some P_i (over the X variables).

Now to compute $C'^{nc} \circ P_i^*$ for each i, we symmetrize the polynomial P_i, the symmetrization happens over the X variables as well as over the Z variables. But in C'^{nc} we are only interested in the monomials (or words) where the rightmost $0.3k$ variables are over Z variables. In the noncommutative circuit C'^{nc}, every sub-word $z_{i_1} z_{i_2} \ldots z_{i_{0.3k}}$ receives a natural ordering $i_1 < i_2 < \ldots < i_{0.3k}$.

Notice that

$$P_i^*(X, Z) = \sum_{\sigma \in S_{1.3k}} \prod_{j=1}^{1.3k} \left(\sum_{\ell:\zeta_i(\ell)=\sigma(j)} x_\ell + z_{\sigma(j)} \right).$$

Our goal is to understand the part of $P_i^*(X, Z)$ where each monomial ends with the sub-word $z_{i_1} z_{i_2} \ldots z_{i_{0.3k}}$ and the top k symbols are over the X variables. For a fixed set of indices $W = \{i_1 < i_2 < \ldots < i_{0.3k}\}$, define the set $T = [1.3k] \setminus W$. Let $S_{[k],T}$ be the set of permutations $\sigma \in S_{1.3k}$ such that $\sigma : [k] \to T$ and $\sigma(k + j) = i_j$ for $1 \le j \le 0.3k$. As we have fixed the last $0.3k$ positions, each $\sigma \in S_{[k],T}$ corresponds to some $\sigma' \in S_k$. Let $Z_W = z_{i_1} z_{i_2} \ldots z_{i_{0.3k}}$. Now the following claim is immediate.

[3] Since the syntactic degree of the circuit is not bounded here, and if we have to account for the bit level complexity (over $\mathbb{Z}$) of the scalars generated in the intermediate stage we may get field elements whose bit level complexity is exponential in the input size. So, a standard technique is to take a random prime of polynomial bit-size and evaluate the circuit modulo that prime.

[4] This is also used in [8].

▷ Claim 21. The part of $P_i^*(X,Z)$ where each monomial ends with the sub-word Z_W and the first k variables are from X, is $P_{i,W}^* \cdot Z_W$, where

$$P_{i,W}^*(X) = \sum_{\sigma\in S_{[k],T}} \prod_{j=1}^{k} \left(\sum_{\ell:\zeta_i(\ell)=\sigma(j)} x_\ell \right) = \sum_{\sigma'\in S_k} \prod_{j=1}^{k} \left(\sum_{\ell:\zeta_i(\ell)=\sigma'(j)} x_\ell \right).$$

Notice that, $\sum_{W\subseteq[1.3k]:|W|=0.3k} P_{i,W}^*$ contains all the *colourful* degree-k multilinear monomials over X. We now obtain the following.

$$(C'^{nc} \circ P_i^*)(X,Z) = \sum_{W\subseteq[1.3k]:|W|=0.3k} \big(C^{nc}(X) \circ P_{i,W}^*(X)\big) \cdot Z_W.$$

Setting each $z_i = 1$ and using distributivity of Hadamard product, we get $(C'^{nc} \circ P_i^*)(X,\vec{1}) = C^{nc}(X) \circ \sum_{W\subseteq[1.3k]:|W|=0.3k} P_{i,W}^*$ which is the *colourful* multilinear part of the input circuit.

We now consider $(C' \circ^s P_i)(X,Z)$ and substitute 1 for each Z variable and do a randomized PIT test on the X variables using Demillo-Lipton-Schwartz-Zippel Lemma [9, 24, 18]. By Lemma 16, for any random $\vec{a} \in \mathbb{F}^n$, $(C' \circ^s P_i)(\vec{a})$ can be computed in $O^*(2^{1.3k}) = O^*(2.46^k)$ time and poly(n,k) space. This suffices to check whether the resulting circuit is identically zero or not. We repeat the procedure for each coloring and obtain a randomized $O^*(4.32^k)$ algorithm. This completes the proof of Theorem 2. ◀

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Fast Exact Algorithms Using Hadamard Product of Polynomials. *CoRR*, abs/1807.04496, 2018. arXiv:1807.04496.

3 Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. Arithmetic Circuits and the Hadamard Product of Polynomials. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 25–36, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2304.

4 Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 677–686, 2010. doi:10.1145/1806689.1806782.

5 Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. *Computational Complexity*, 27(1):1–29, 2018. doi:10.1007/s00037-016-0148-5.

6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting Paths and Packings in Halves. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, pages 578–586, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Evaluation of permanents in rings and semirings. *Inf. Process. Lett.*, 110(20):867–870, 2010. doi:10.1016/j.ipl.2010.07.005.

8 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164, 2018. doi:10.1145/3188745.3188902.

9 Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

11 Ismor Fischer. Sums of Like Powers of Multivariate Linear Forms. *Mathematics Magazine*, 67(1):59–61, 1994. doi:10.1080/0025570X.1994.11996185.

12 Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm Engineering for Color-Coding with Applications to Signaling Pathway Detection. *Algorithmica*, 52(2):114–132, 2008. doi:10.1007/s00453-007-9008-7.

13 Ioannis Koutis. Faster Algebraic Algorithms for Path and Packing Problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 575–586, 2008. doi:10.1007/978-3-540-70575-8_47.

14 Ioannis Koutis and Ryan Williams. LIMITS and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3):31:1–31:18, 2016. doi:10.1145/2885499.

15 Hwangrae Lee. Power Sum Decompositions of Elementary Symmetric Polynomials. In *Linear Algebra and its Applications*, volume 492. Elsevier, August 2015.

16 Kevin Pratt. Faster Algorithms via Waring Decompositions. *CoRR*, abs/1807.06194, 2018. arXiv:1807.06194.

17 H.J. Ryser. *Combinatorial mathematics*. Carus mathematical monographs. Mathematical Association of America; distributed by Wiley [New York, 1963. URL: https://books.google.co.in/books?id=wOruAAAAMAAJ.

18 Jacob T. Schwartz. Fast Probabilistic algorithm for verification of polynomial identities. *J. ACM.*, 27(4):701–717, 1980.

19 Amir Shpilka and Amir Yehudayoff. Arithmetic Circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.

20 Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast Parallel Computation of Polynomials Using Few Processors. *SIAM J. Comput.*, 12(4):641–644, 1983. doi:10.1137/0212043.

21 Richard Ryan Williams. The Polynomial Method in Circuit Complexity Applied to Algorithm Design (Invited Talk). In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 47–60, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.47.

22 Ryan Williams. Finding paths of length k in $\mathrm{O}^*(2^\mathrm{k})$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009. doi:10.1016/j.ipl.2008.11.004.

23 Ryan Williams. Algorithms for Circuits and Circuits for Algorithms. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 248–261, 2014. doi:10.1109/CCC.2014.33.

24 R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. of the Int. Sym. on Symbolic and Algebraic Computation*, pages 216–226, 1979.

Approximate Online Pattern Matching in Sublinear Time

Diptarka Chakraborty
National University of Singapore, Singapore
diptarka@comp.nus.edu.sg

Debarati Das
University of Copenhagen, Denmark
debaratix710@gmail.com

Michal Koucký
Computer Science Institute of Charles University, Czech Republic
koucky@iuuk.mff.cuni.cz

Abstract

We consider the approximate pattern matching problem under edit distance. In this problem we are given a pattern P of length m and a text T of length n over some alphabet Σ, and a positive integer k. The goal is to find all the positions j in T such that there is a substring of T ending at j which has edit distance at most k from the pattern P. Recall, the edit distance between two strings is the minimum number of character insertions, deletions, and substitutions required to transform one string into the other. For a position t in $\{1, ..., n\}$, let k_t be the smallest edit distance between P and any substring of T ending at t. In this paper we give a constant factor approximation to the sequence $k_1, k_2, ..., k_n$. We consider both offline and online settings.

In the offline setting, where both P and T are available, we present an algorithm that for all t in $\{1, ..., n\}$, computes the value of k_t approximately within a constant factor. The worst case running time of our algorithm is $\tilde{O}(nm^{3/4})$.

In the online setting, we are given P and then T arrives one symbol at a time. We design an algorithm that upon arrival of the t-th symbol of T computes k_t approximately within $O(1)$-multiplicative factor and $m^{8/9}$-additive error. Our algorithm takes $\tilde{O}(m^{1-(7/54)})$ amortized time per symbol arrival and takes $\tilde{O}(m^{1-(1/54)})$ additional space apart from storing the pattern P. Both of our algorithms are randomized and produce correct answer with high probability. To the best of our knowledge this is the first algorithm that takes worst-case sublinear (in the length of the pattern) time and sublinear extra space for the online approximate pattern matching problem. To get our result we build on the technique of Chakraborty, Das, Goldenberg, Koucký and Saks [FOCS'18] for computing a constant factor approximation of edit distance in sub-quadratic time.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Approximate Pattern Matching, Online Pattern Matching, Edit Distance, Sublinear Algorithm, Streaming Algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.10

Related Version A full version of the paper is available at `https://arxiv.org/abs/1810.03664`.

Funding The research leading to these results is partially supported by the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X and by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787.

Acknowledgements Authors would like to thank anonymous reviewers for many helpful suggestions and comments on an earlier version of this paper.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 10; pp. 10:1–10:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Finding the occurrences of a pattern in a larger text is one of the fundamental problems in computer science. Due to its immense applications this problem has been studied extensively under several variations [25, 21, 5, 17, 23, 18, 24, 31, 26]. One of the most natural variations is where we are allowed to have a small number of errors while matching the pattern. This problem of pattern matching while allowing errors is known as *approximate pattern matching*. The kind of possible errors varies with the applications. Generally we capture the amount of errors by the metric defined over the set of strings. One common and widely used distance measure is the edit distance (aka *Levenshtein distance*) [28]. The edit distance between two strings T and P denoted by $d_{\text{edit}}(T, P)$ is the minimum number of character insertions, deletions, and substitutions required to transform one string into the other. In this paper we focus on the approximate pattern matching problem under edit distance. This problem has various applications ranging from computational biology, signal transmission, web searching, text processing to many more.

Given a pattern P of length m and a text T of length n over some alphabet Σ, and an integer k we want to identify all the substrings of T at edit distance at most k from P. As the number of such substrings might be quadratic in n and one wants to obtain efficient algorithms, one focuses on finding the set of all right-end positions in T of those substrings at distance at most k. More specifically, for a position t in T, we let k_t be the smallest edit distance of a substring of T ending at t-th position in T. (We number positions in T and P from 1.) The goal is to compute the sequence $k_1, k_2, \ldots, k_n$ for P and T. Using basic dynamic programming paradigm we can solve this problem in $O(nm)$ time [33]. Later Masek and Paterson [29] shaved a $\log n$ factor from the above running time bound. Despite of a long line of research, this running time remains the best till now. Recently, Backurs and Indyk [8] indicate that this $O(nm)$ bound cannot be improved significantly unless the Strong Exponential Time Hypothesis (SETH) is false. Moreover Abboud et al. [3] showed that even shaving an arbitrarily large polylog factor would imply that NEXP does not have non-uniform NC^1 circuits which is likely but hard to prove conclusion. More hardness results can be found in [2, 9, 1, 4].

In this paper we focus on finding an approximation to the sequence $k_1, k_2, \ldots, k_n$ for P and T. For reals $c, a \geq 0$, a sequence $\tilde{k}_1, \ldots, \tilde{k}_n$ is (c, a)-approximation to $k_1, \ldots, k_n$, if for each $t \in \{1, \ldots, n\}$, $k_t \leq \tilde{k}_t \leq c \cdot k_t + a$. Hence, c is the multiplicative error and a is the additive error of the approximation. An algorithm computes (c, a)-approximation to approximate pattern matching if it outputs a (c, a)-approximation of the true sequence $k_1, k_1, \ldots, k_n$ for P and T. We refer to $(c, 0)$-approximation simply as c-approximation. Our main theorem is the following.

▶ **Theorem 1.** *There is a constant $c \geq 1$ and there is a randomized algorithm that computes a c-approximation to approximate pattern matching in time $\tilde{O}(n \cdot m^{3/4})$ with probability at least $(1 - 1/n^3)$.*

In the recent past researchers also studied the approximate pattern matching problem in the online setting. The online version of this pattern matching problem mostly arises in real life applications that require matching pattern in a massive data set, like in telecommunications, monitoring Internet traffic, building firewall to block viruses and malware connections and many more. The online approximate pattern matching is as follows: we are given a pattern P first, and then the text T is coming symbol by symbol. Upon receipt of the t-th symbol we should output the corresponding k_t. The online algorithm runs in *amortized time* $O(\ell)$ if it runs in total time $O(n \cdot \ell)$. We also say that the online algorithm uses *extra space* $O(s)$ if in addition to storing the pattern P it uses at most $O(s)$ cells of memory at any time.

▶ **Theorem 2.** *There is a constant $c \geq 1$ and there is a randomized online algorithm that computes $(c, m^{8/9})$-approximation to approximate pattern matching in amortized time $\tilde{O}(m^{1-(7/54)})$ and extra space $\tilde{O}(m^{1-(1/54)})$ with probability at least $1 - 1/poly(m)$.*

To the best of our knowledge this is the first online approximation algorithm that takes sublinear (in the length of the pattern) running time and sublinear extra space for the approximate pattern matching problem. Designing algorithm that uses small extra space is quite natural from the practical point of view and has been considered for many problems including pattern matching, e.g. [32, 22].

To prove our result we use the technique developed by Chakraborty, Das, Goldenberg, Koucký and Saks in [10, 11], where they provide a sub-quadratic time constant factor approximation algorithm for the edit distance problem. In particular, in [11] authors describe a constant factor approximation algorithm that given two strings of length n runs in time $\tilde{O}(n^{12/7})$. Now suppose we only have a black-box access to that approximation algorithm for computing the edit distance. Then we claim that we get $O(1)$-approximation to the offline approximate pattern matching problem in time $\tilde{O}(nm^{6/7})$. Let us first set a parameter $k = m^{6/7}$. Now in the first phase we run $O(nk)$ time algorithm by Landau and Vishkin [27] and get all the values of k_t which are at most k. In the next phase we divide the text T into overlapping substrings of length m with overlap of $m - k$. In other words for every t that are multiple of k consider the substring $T_{t-m+1,t}$ (that starts at $(t - m + 1)$-th symbol and ends at t-th symbol). For all the positions t that are multiple of k and $k_t > k$ (as identified by the first phase) we use the edit distance algorithm of [11] to get $O(1)$-approximation of $d_{\text{edit}}(T_{t-m+1,t}, P)$ and output that value as $\tilde{k}_t$. Since $k_t \leq d_{\text{edit}}(T_{t-m+1,t}, P) \leq 2k_t$, the output $\tilde{k}_t$ is an $O(1)$-approximation of k_t. For all the remaining values of t (that are not multiples of k, and $k_t > k$) we output $\tilde{k}_{t'} + (t - t')$ where $t' = \lfloor \frac{t}{k} \rfloor \cdot k$, as an estimate of k_t. Since $k_{t'} - (t - t') \leq k_t \leq k_{t'} + (t - t')$, for all t such that $k_t > k$ we get $O(1)$-approximation of k_t. Note, the above described process takes $\tilde{O}(nm^{6/7})$ time and thus breaks $O(nm)$ barrier for the offline approximate pattern matching problem for constant factor approximation. However our claimed running time in Theorem 1 is better than that of this black-box algorithm.

In this paper we first design an offline algorithm by building upon the technique used in [11]. To do this we exploit the similarity between the "dynamic programming graphs" (see Section 2) for approximate pattern matching problem and the edit distance problem. To get $\tilde{O}(nm^{3/4})$ time algorithm for the offline approximate pattern matching problem still requires careful modifications to the edit distance algorithm. However the scenario becomes much more involved if one wants to design an online algorithm using only a small amount of extra space. The approximation algorithm for edit distance in [11] works in two phases: first a covering algorithm is used to discover a suitable set of shortcuts in the pattern matching graph, and then a min-cost path algorithm on a grid graph with the shortcuts yields the desired result. In the online setting we carefully interleave all of the above phases. However that by itself is not sufficient since the first phase, i.e., the covering algorithm used in [11] essentially relies on the fact that both of the strings are available at any point of time. We modify the covering technique so that it can also be implemented in the situation when we cannot see the full text. We show that if we store the pattern P then we need only $O(m^{1-\gamma})$ extra space (for some small constant $\gamma > 0$) to perform the sampling. Furthermore, the min-cost path algorithm in [11] takes $O(m)$ space. We modify that algorithm too in a way so that it also works using only $O(m^{1-\gamma})$ space (for some small constant $\gamma > 0$). We describe our algorithm in more details in Section 5.

1.1 Related work

The approximate pattern matching problem is one of the most extensively studied problems in modern computer science due to its direct applicability to data driven applications. In contrast to the exact pattern matching here a text location has a match if the distance between the pattern and the text is within some tolerated limit. In our work we study the approximate pattern matching under edit distance metric. The very first $O(nm)$-time algorithm was given by Sellers [33] in 1980. Masek and Paterson [29] proposed an $O(nm/\log n)$-time $O(n)$-space algorithm using Four Russians [7] technique. Later [30, 27, 20] gave $O(nk)$-time algorithms where k is the upper limit of allowed edit operations. All of these algorithms use either $O(m^2)$ or $O(n)$ space. However [19, 36] note that achieving $O(m)$ space is also possible while maintaining the running time. A faster (for small values of k) algorithm was given by Cole and Hariharan [16], which has running time $O(n(1 + k^4/m))$. We refer the interested readers to a beautiful survey by Navarro [31] for a comprehensive treatment on this topic. We have already seen in the previous section that any c-approximation algorithm for the edit distance problem can be transformed into an $O(c)$-approximation algorithm for the approximate pattern matching problem. We get $(\log m)^{O(1/\epsilon)}$-approximation to the approximate pattern matching problem in time $O(nm^{\frac{1}{2}+\epsilon})$ (for every $\epsilon > 0$) from the edit distance algorithm of Andoni *et al.* [6]. To achieve the same running time while having only constant factor approximation is an important open problem.

All the above mentioned algorithms assume that the entire text is available from the very beginning of the process. However in the online version, the pattern is given at the beginning and the text arrives in a stream, one symbol at a time. Clifford *et al.* [12] gave a "black-box algorithm" for online approximate matching where the supported distance metrics are Hamming distance, matching with wildcards, L_1 and L_2 norm. Their algorithm has running time $O(\sum_{j=1}^{\log_2 m} T(n, 2^{j-1})/n)$ per symbol arrival, where $T(n, m)$ is the running time of the best offline algorithm. This result was extended in [14] by introducing an algorithm solving online approximate pattern matching under edit distance metric in time $O(k \log m)$ per symbol arrival. This algorithm uses $O(m)$-space. In [15] the running time was further improved to $O(k)$ per symbol. However none of these algorithms for edit distance metric is black-box and they highly depend on the specific structure of the corresponding offline algorithm. Recently, Starikovskaya [34] gave a randomized algorithm which has a worst case time complexity of $O((k^2\sqrt{m} + k^{13})\log^4 m)$ and uses space $O(k^8\sqrt{m}\log^6 m)$. Although her algorithm takes both sublinear time and sublinear space for small values of k, heavy dependency on k in the complexity terms makes it much worse than the previously known algorithms in the high regime of k. On the lower bound side, Clifford, Jalsenius and Sach [13] showed in the *cell-probe model* that expected amortized running time of any randomized algorithm solving online approximate pattern matching problem must be $\Omega(\sqrt{\log m}/(\log\log m)^{3/2})$ per output.

2 Preliminaries

We recall some basic definitions of [11]. Consider the text T of length n to be aligned along the horizontal axis and the pattern P of length m to be aligned along the vertical axis. For $i \in \{1, \ldots, n\}$, T_i denotes the i-th symbol of T and for $j \in \{1, \ldots, m\}$, P_j denotes the j-th symbol of P. $T_{s,t}$ is the substring of T starting by the s-th symbol and ending by the t-th symbol of T. For any interval $I \subseteq \{0, \ldots, n\}$, T_I denotes the substring of T indexed by $I \setminus \{\min(I)\}$ and for $J \subseteq \{0, \ldots, m\}$, P_J denotes the substring of P indexed by $J \setminus \{\min(J)\}$.

Edit distance and pattern matching graphs

For a text T of length n and a pattern P of length m, the *edit distance graph* $G_{T,P}$ is a directed weighted graph called a grid graph with vertex set $\{0, \cdots, n\} \times \{0, \cdots, m\}$ and following three types of edges: $(i-1, j) \to (i, j)$ (H-steps), $(i, j-1) \to (i, j)$ (V-steps) and $(i-1, j-1) \to (i, j)$ (D-steps). Each H-step or V-step has cost 1 and each D-step costs 0 if $T_i = P_j$ and 1 otherwise. The *pattern matching graph* $\tilde{G}_{T,P}$ is the same as the edit distance graph $G_{T,P}$ except for the cost of horizontal edges $(i, 0) \to (i+1, 0)$ which is zero.

For intervals $I \subseteq \{0, \ldots, n\}$ and $J \subseteq \{0, \ldots, m\}$, $G_{T,P}(I \times J)$ is the subgraph of $G_{T,P}$ induced on $I \times J$. Clearly, $G_{T,P}(I \times J) \cong G_{T_I, P_J}$. We define the cost of a path τ in G_{T_I,P_J}, denoted by $\mathrm{cost}_{G_{T_I,P_J}}(\tau)$, as the sum of the costs of its edges. We also define the cost of a graph G_{T_I,P_J}, denoted by $\mathrm{cost}(G_{T_I,P_J})$, as the cost of the cheapest path from $(\min I, \min J)$ to $(\max I, \max J)$.

The following is well known in the literature (e.g. see [33]).

▶ **Proposition 3.** *Consider a pattern P of length m and a text T of length n, and let $G = \tilde{G}_{T,P}$. For any $t \in \{1, \ldots, n\}$, let $I = \{0, \cdots, t\}$, and $J = \{0, \cdots, m\}$. Then $k_t = cost(G(I \times J)) = \min_{i \le t} d_{edit}(T_{i,t}, P)$.*

A similar proposition is also true for the edit distance graph.

▶ **Proposition 4.** *Consider a pattern P of length m and a text T of length n, and let $G = G_{T,P}$. For any $i_1 \le i_2 \in \{1, \cdots, n\}$, $j_1 \le j_2 \in \{1, \cdots, m\}$ let $I = \{i_1 - 1, \cdots, i_2\}$ and $J = \{j_1 - 1, \cdots, j_2\}$. Then $cost(G(I \times J)) = d_{edit}(T_{i_1,i_2}, P_{j_1,j_2})$.*

Let G be a grid graph on $I \times J$ and $\tau = (i_1, j_1), \ldots, (i_l, j_l)$ be a path in G. *Horizontal projection* of a path τ is the set $\{i_1, \ldots, i_l\}$. Let I' be an interval contained in the horizontal projection of τ, then $\tau_{I'}$ denotes the (unique) minimal subpath of τ with horizontal projection I'. Let $G' = G(I' \times J')$ be a subgraph of G. For $\delta \ge 0$ we say that $I' \times J'$ $(1-\delta)$-*covers* the path τ if the initial and the final vertex of $\tau_{I'}$ are at a vertical distance of at most $\delta(|I'| - 1)$ from $(\min(I'), \min(J'))$ and $(\max(I'), \max(J'))$, resp..

A *certified box* of G is a pair $(I' \times J', \ell)$ where $I' \subseteq I$, $J' \subseteq J$ are intervals, and $\ell \in \mathbb{N} \cup \{0\}$ such that $\mathrm{cost}(G(I' \times J')) \le \ell$. At high level, our goal is to approximate each path τ in G by a path via the corner vertices of certified boxes. For that we want that a substantial portion of the path τ goes via those boxes and that the sum of the costs of the certified boxes is not much larger than the actual cost of the path. The next definition makes our requirements precise. Let $\sigma = \{(I_1 \times J_1, \ell_1), (I_2 \times J_2, \ell_2), \ldots, (I_s \times J_s, \ell_s)\}$ be a sequence of certified boxes in G. Let τ be a path in $G(I \times J)$ with horizontal projection I. For any $k, \zeta \ge 0$, we say that σ (k, ζ)-*approximates* τ if the following three conditions hold:

1. $I_1, \ldots, I_s$ is a decomposition of I, i.e., $I = \bigcup_{i \in [s]} I_i$, and for all $i \in [s-1]$, $\min(I_{i+1}) = \max(I_i)$.
2. For each $i \in [s]$, $I_i \times J_i$ $(1 - \ell_i/(|I_i| - 1))$-covers τ.
3. $\sum_{i \in [s]} \ell_i \le k \cdot \mathrm{cost}(\tau) + \zeta$.

3 Offline approximate pattern matching

3.1 Technical Overview

To prove Theorem 1 we design an algorithm as follows. For $k = 2^j$, $j = 0, \ldots, \log m^{3/4}$, we run the standard $O(nk)$ algorithm by Landau and Vishkin [27] to identify all t such that $k_t \le k$. To identify positions with $k_t \le k$ for $k > m^{3/4}$ where k is a power of two we will use the technique of [11] to compute $(O(1), O(m^{3/4}))$-approximation of $k_1, \ldots, k_n$. The obtained information can be combined in a straightforward manner to get a single $O(1)$-approximation to $k_1, \ldots, k_n$: For each t, if for some $2^j \leq m^{3/4}$, k_t is at most 2^j (as determined by the former algorithm) then output the exact value of k_t using the algorithm of [27], otherwise output the approximation of k_t found by the latter algorithm. This way, for $k_t \leq m^{3/4}$ we will get the exact value, and for $k > m^{3/4}$ we will get an $O(1)$-approximation. We will now elaborate on the latter algorithm based on [11]. The edit distance algorithm of [11] has two phases which we will also use. The first phase (*covering phase*) identifies a set of *certified boxes*, subgraphs of the pattern matching graph with good upper bounds on their cost. These certified boxes should cover the min-cost paths of interest. Then the next phase runs a min-cost path algorithm on these boxes to obtain the output sequence. We will show that both of these phases take $\widetilde{O}(nm^{3/4})$ time and so the overall running time will be $\widetilde{O}(nm^{3/4})$.

We next describe the two phases of the algorithm. The algorithm will use the following parameters: $w_1 = m^{1/4}$, $w_2 = m^{1/2}$, $d = m^{1/4}$, $\theta = m^{-1/4}$. The meaning of the parameters is essentially the same as in [11] though their setting is different. Let $c_0, c_1 \geq 0$ be the large enough constants from [11]. For simplicity we will assume without loss of generality that w_1 and w_2 are powers of two (by rounding them down to the nearest powers of two), θ is a reciprocal of a power of two (by decreasing θ by at most a factor of two), $w_2|m$ (by chopping off a small suffix from P which will affect the approximation by a negligible additive error as $m^{3/4} \gg w_2$), and $m|n$ (if not we can run the algorithm twice: on the largest prefix of T of length divisible by m and then on the largest suffix of T of length divisible by m). The algorithm will not explicitly compute k_t for all t but only for t where t is a multiple of w_2, and then it will use the same value for each block of w_2 consecutive k_t's. Again, this will affect the approximation by a negligible additive error.

3.2 Covering phase

We describe the first phase of the algorithm now. First, we partition the text T into substrings $T_1^0, \ldots, T_{n_0}^0$ of length m, where $n_0 = n/m$. Then we process each of the parts independently. Let T' be one of the parts. We partition T' into substrings $T_1^1, T_2^1, \ldots, T_{n_1}^1$ of length w_1, and we also partition T' into substrings $T_1^2, T_2^2, \ldots, T_{n_2}^2$ of length w_2, where $n_1 = m/w_1$ and $n_2 = m/w_2$. For a substring u of v starting by i-th symbol of v and ending by j-th symbol of v, we let $\{i-1, i, i+1, \ldots, j\}$ be its *span.* Moreover for $\delta \in (0,1)$ we call u to be (δ)-aligned if both $i-1$ and $j-1$ are divisible by $\delta(j-i)$. The covering algorithm proceeds in phases $j = 0, \ldots, \lceil \log 1/\theta \rceil$ associated with $\epsilon_j = 2^{-j}$. Similar to the edit distance algorithm, here also each phase has two parts, namely the *dense substrings* and the *extension sampling.* Then the covering algorithm proceeds as follows:

Dense substrings

In this part the algorithm aims to identify for each ϵ_j, a set of substrings T_i^1 that are similar (i.e., up to "small" edit distance) to more than d $(\epsilon_j/8)$-aligned, w_1 length substrings of P. We identify each T_i^1 by testing a random sample of relevant substrings of P. If we determine with high confidence that there are at least $\Omega(d)$ substrings of P similar to T_i^1, we add T_i^1 into a set D_j of such strings, and we also identify all $T_{i'}^1$ that are similar to T_i^1. By triangle inequality we would also expect them to be similar to many relevant substrings of P. So we add these $T_{i'}^1$ to D_j as well as we will not need to process them anymore. We output the set of certified boxes of edit distance $O(\epsilon_j w_1)$ found this way. More formally:

For $j = \lceil \log 1/\theta \rceil, \ldots, 0$, the algorithm maintains sets D_j of substrings T_i^1. These sets are initially empty.

Step 1. For each $i = 1, \ldots, n_1$ and $j = \lceil \log 1/\theta \rceil, \ldots, 0$, if T_i^1 is in D_j then we continue with the next i and j. Otherwise we process it as follows.

Step 2. Set $\epsilon_j = 2^{-j}$. Independently at random, sample $8c_0 \cdot m \cdot (\epsilon_j w_1 d)^{-1} \cdot \log n$ many $(\epsilon_j/8)$-aligned substrings of P of length w_1. For each sampled substring u check if its edit distance from T_i^1 is at most $\epsilon_j w_1$. If less than $\frac{1}{2} \cdot c_0 \cdot \log n$ of the samples have their edit distance from T_i^1 below $\epsilon_j w_1$ then we are done with processing this i and j and we continue with the next pair.

Step 3. Otherwise we identify all substrings $T_{i'}^1$ that are not in D_j and are at edit distance at most $2\epsilon_j w_1$ from T_i^1, and we let X to be the set of their spans relative to the whole T.

Step 4. Then we identify all $(\epsilon_j/8)$-aligned substrings of P of length w_1 that are at edit distance at most $3\epsilon_j w_1$ from T_i^1, and we let Y to be the set of their spans. (We also allow some $(\epsilon_j/8)$-aligned substrings of P of edit distance at most $6\epsilon_j w_1$ to be included in the set Y as some might be misidentified to have the smaller edit distance from T_i^1 by our procedure that searches for them, see further.)

Step 5. For each pair of spans (I, J) from $X \times Y$ we output corresponding certified box $(I \times J, 8\epsilon_j w_1)$. We add substrings corresponding to X into D_j and continue with the next pair i and j.

Once we process all pairs of i and j, we proceed to the next phase: *extension sampling*.

Extension sampling

In this part for every $\epsilon_j = 2^{-j}$ and every substring T_i^2, which does not have all its substrings T_ℓ^1 contained in D_j we randomly sample a set of such T_ℓ^1's. For each sampled T_ℓ^1 we determine all relevant substrings of P at edit distance at most $\epsilon_j w_1$ from T_ℓ^1. There should be $O(d)$-many such substrings of P. We extend each such substring into a substring of size $|T_i^2|$ within P and we check the edit distance of the extended string from T_i^2. For each extended substring of edit distance at most $3\epsilon_j w_2$ we output a set of certified boxes.

Here we define the appropriate extension of substrings. Let u be a substring of T of length less than $|P|$, and let v be a substring of u starting by the i-th symbol of u. Let v' be a substring of P of the same length as v starting by the j-th symbol of P. The *diagonal extension u' of v' in P with respect to u and v*, is the substring of P of length $|u|$ starting at position $j - i$. If $(j - i) \leq 0$ then the extension u' is the prefix of P of length $|u|$, and if $j - i + |P| > |P|$ then the extension u' is the suffix of P of length $|u|$.

Step 6. Process all pairs $i = 1, \ldots, n_2$ and $j = \lceil \log 1/\theta \rceil, \ldots, 0$.

Step 7. Independently at random, sample $c_1 \cdot \log^2 n \cdot \log m$ substrings T_ℓ^1 that are part of T_i^2 and that are not in D_j. (If there is no such substring continue for the next pair of i and j.)

Step 8. For each T_ℓ^1, find all $(\epsilon_j/8)$-aligned substrings v' of P of length w_1 that are at edit distance at most $\epsilon_j w_1$ from T_ℓ^1.

Step 9. For each v' determine its diagonal extension u' with respect to T_i^2 and T_ℓ^1. Check if the edit distance of u' and T_i^2 is less than $3\epsilon_j w_2$. If so, compute it and denote the distance by c. Let I' be the span of T_i^2 relative to T, and J' be the span of u' in P. For all powers a and b of two, $m^{3/4} \le a \le b \le m$, output the certified box $(I' \times J', c + a + b)$. Proceed for the next i and j.

This ends the covering algorithm which outputs various certified boxes.

To implement the above algorithm we will use Ukkonen's [35] $O(nk)$-time algorithm to check whether the edit distance of two strings of length w_1 is at most $\epsilon_j w_1$ in time $O(w_1^2 \epsilon_j)$. Given the edit distance is within this threshold the algorithm can also output its precise value. We use this algorithm in Step 3. To identify all substrings of length w_1 at edit distance at most $\epsilon_j w_1$ of S from a given string R (where S is the pattern P of length m and R is one of the T_i^1 of length w_1), in Step 4, we use the $O(nk)$-time pattern matching algorithm of Landau and Vishkin [27]. For a given threshold k, this algorithm determines for each position t in S, whether there is a substring of edit distance at most k from R ending at that position in S. If the algorithm reports such a position t then we know by the following proposition that the substring $S_{t-|R|+1,t}$ is at edit distance at most $2k$. At the same time we are guaranteed to identify all the substrings of S of length w_1 at edit distance at most k from R. Hence in Step 4, finding all the substrings at distance $3\epsilon_j w_1$ with perhaps some extra substrings of edit distance at most $6\epsilon_j w_1$ can be done in time $O(m w_1 \epsilon_j)$.

▶ **Proposition 5.** *For strings S and R, and integers $t \in \{1, \ldots, |S|\}$, $k \ge 0$, if $\min_{i \le t} d_{edit}(S_{i,t}, R) \le k$ then $d_{edit}(S_{t-|R|+1,t}, R) \le 2k$.*

Proof. Let $S_{i,t}$ be the best match for R ending by the t-th symbol of S. Hence, $k = d_{\text{edit}}(S_{i,t}, R)$. If $S_{i,t}$ is by ℓ symbols longer than R then $k \ge \ell$ and $d_{\text{edit}}(S_{t-|R|+1,t}, R) \le k + \ell \le 2k$ by the triangle inequality. The same is true if $S_{i,t}$ is shorter by ℓ symbols. ◀

3.3 Correctness of the covering algorithm

▶ **Lemma 6.** *Let $t \ge 1$ be such that t is a multiple of w_2. Let τ_t be the min-cost path between vertex $(t - m, 0)$ and (t, m) in the edit distance graph $G = G_{T,P}$ of T and P of cost at least $m^{3/4} \ge \theta m$. The covering algorithm outputs a set of weighted boxes $\mathcal{R}$ such that every $(I \times J, \ell) \in \mathcal{R}$ is correctly certified i.e., cost$(G(I \times J)) \le \ell$ and there is a subset of $\mathcal{R}$ that $(O(1), O(k_t))$-approximates τ_t with probability at least $1 - 1/n^7$.*

It is clear from the description of the covering algorithm that it outputs only correct certified boxes from the edit distance graph of T and P, that is for each box $(I \times J, \ell)$, $\text{cost}(G(I \times J)) \le \ell$.

The cost of τ_t corresponds to the edit distance between P and $T_{t-m+1,t}$ and it is bounded by $2k_t$ by Proposition 5. Let k'_t be the smallest power of two $\ge k_t$. We claim that by essentially the same argument as in Proposition 3.8 and Theorem 3.9 of [11] the algorithm outputs with high probability a set of certified boxes that $(O(1), O(k'_t))$-approximates τ_t. Therefore instead of repeating the whole proof, here we sketch the differences between the current covering algorithm with that of [11] and argue about how to handle them.

The main substantial difference is that the algorithm in [11] searches for certified boxes located only within $O(k_t)$ diagonals along the main diagonal of the edit distance graph. (This rests on the observation of Ukkonen [35] that a path of cost $\le k_t$ must pass only through vertices on those diagonals.) Here we process certified boxes in the whole matrix as each t requires a different "main" diagonal. Except for this difference and the order of processing various pieces the algorithms are the same.

The discovery of certified boxes depends on the number (*density*) of relevant substrings of P similar to a given T_i^1. In the edit distance algorithm in [11] this density is measured only in the $O(k_t)$-width strip along the main diagonal of the edit distance graphs whereas here it is measured within the whole P. (So the actual classification of substrings T_i^1 on *dense* (in D_j) and *sparse* (not in D_j) might differ between the two algorithms.) Hence, one could think (though technically not quite correct) that the certified boxes output by the current algorithm form a superset of boxes output by the edit distance algorithm of [11]. However, this difference is immaterial for the correctness argument in Theorem 3.9 of [11].

Another difference is that in Step 4 we use $O(mw_1\epsilon_j)$-time algorithm to search for all the similar substrings. This algorithm will report all the substrings we were looking for and additionally it might report some substrings of up to twice the required edit distance. This necessitates the upper bound $8\epsilon_j w_1$ in certified boxes in Step 5. It also means a loss of factor of at most two in the approximation guarantee as the boxes of interest are reported with the cost $8\epsilon_j w_1$ instead of the more accurate $5\epsilon_j w_1$ of the original algorithm in [11] which would give a $(45, 15\text{cost}(\tau_t))$-approximation. (In that theorem θm represents an (arbitrary) upper bound on the cost of τ_t provided it satisfies certain technical conditions requiring that θ is large enough relative to m. This is satisfied by requiring that $\text{cost}(\tau_t) \geq m^{3/4} \geq \theta m$.)

Another technical difference is that the path τ_t might pass through two edit distance graphs $G_{T^0_{\ell-1},P}$ and $G_{T^0_\ell,P}$, where $t \in [(\ell-1)m+1, \ell m]$. This means that one needs to argue separately about restriction of τ_t to $G_{T^0_{\ell-1},P}$ and $G_{T^0_\ell,P}$. However, the proof of Theorem 3.9 in [11] analyses approximation of the path in separate parts restricted to substrings of T of size w_2. As both t and m are multiples of w_2, the argument for each piece applies in our setting as well.

3.4 Time complexity of the covering algorithm

By analyzing the running time we get the following.

▶ **Lemma 7.** *The covering algorithm runs in time $\widetilde{O}(nm^{3/4})$ with probability at least $1-1/n^8$.*

We analyse the running time of the covering algorithm for each $T' = T_i^0$ separately. We claim that the running time on T' is $\widetilde{O}(m^{7/4})$ so the total running time is $\widetilde{O}((n/m)m^{7/4}) = \widetilde{O}(nm^{3/4})$.

In Step 1, for every $i = 1, \ldots, n_1$ and $j = 0, \ldots, \log m^{1/4}$, we might sample $O(\frac{m}{\epsilon_j w_1 d} \cdot \log n)$ substrings of P of length w_1 and check whether their edit distance from T_i^1 is at most $\epsilon_j w_1$. This takes time at most $\widetilde{O}(\frac{m}{\epsilon_j w_1 d} \cdot \frac{m}{w_1} \cdot w_1^2 \epsilon_j) = \widetilde{O}(m^2/d) = \widetilde{O}(m^{7/4})$ in total.

We say that a bad event happens either if some substring T_i^1 has more than d relevant substrings of P having distance at most $\epsilon_j w_1$ but we sample less than $\frac{1}{2} \cdot c_0 \log n$ of them, or if some substring T_i^1 has less than $d/4$ relevant substrings of P having distance at most $\epsilon_j w_1$ but we sample more than $\frac{1}{2} \cdot c_0 \log n$ of them. By Chernoff bound, the probability of a bad event happening during the whole run of the covering algorithm is bounded by $\exp(-O(\log n)) \leq 1/n^8$, for sufficiently large constant c_0. Assuming no bad event happens we analyze the running time of the algorithm further.

Each substring T_i^1 that reaches Step 3 can be associated with a set of its relevant substrings in P of edit distance at most $\epsilon_j w_1$ from it. The number of these substrings is at least $d/4$ many. These substrings must be different for different strings T_i^1 that reach Step 3 as if they were not distinct then the two substrings T_i^1 and $T_{i'}^1$ would be at edit distance at most $2\epsilon_j w_1$ from each other and one of them would be put into D_j in Step 5 while processing the other one so it could not reach Step 3. Hence, we can reach Steps 3–5 for at most $\frac{8m}{\epsilon_j w_1} \cdot \frac{4}{d}$ strings T_i^1. For a given j and each T_i^1 that reaches Step 3, the execution of Steps 3 and 4 takes $O(mw_1\epsilon_j)$ time, hence we will spend in them $\widetilde{O}(m^2/d) = \widetilde{O}(m^{7/4})$ time in total.

Step 5 can report for each j at most $\frac{8m}{\epsilon_j w_1} \cdot \frac{m}{w_1}$ certified boxes, so the total time spent in this step is $\widetilde{O}(m^2/w_1) = \widetilde{O}(m^{7/4})$ as $\epsilon_j w_1 \geq 1/4$.

Step 7 takes order less time than Step 8. In Step 8 we use Ukkonen's [35] $O(nk)$-time edit distance algorithm to check the distance of strings of length w_1. We need to check $\widetilde{O}(n_2 \cdot \frac{m}{\epsilon_j w_1})$ pairs for the total cost $\widetilde{O}(\frac{m}{w_2} \cdot \frac{m}{\epsilon_j w_1} \cdot w_1^2 \epsilon_j) = \widetilde{O}(m^{7/4})$ per j.

As no bad event happens, for each T^1_ℓ, there will be at most $d/4$ strings v' processed in Step 9. We will spend $O(w_2^2 \epsilon_j)$ time on each of them to check for edit distance and $O(\log^2 n)$ to output the certified boxes. Hence, for each j we will spend here $\widetilde{O}(\frac{m}{w_2} \cdot d w_2^2 \epsilon_j)$ time, which is $\widetilde{O}(m w_2 d)$ in total.

Thus, the total time spent by the algorithm in each of the steps is $\widetilde{O}(m^{7/4})$ as required.

4 Min-cost Path in a Grid Graph with Shortcuts

In this section we explain how we use certified boxes to calculate the approximation of k_t's. Consider any grid graph G. A *shortcut* in G is an additional edge $(i, j) \to (i', j')$ with cost ℓ, where $i < i'$ and $j < j'$.

Let $G_{T,P}$ be the edit distance graph for T and P. Let $(I \times J, \ell)$ be a certified box in $G_{T,P}$ with $|I| = |J|$. If $\ell < 1/2(|I| - 1)$ add a shortcut edge $e_{I,J}$ from vertex $(\min I, \min J + \ell)$ to vertex $(\max I, \max J - \ell)$ with cost 3ℓ. Do this for all certified boxes output by the covering algorithm to obtain a graph $G'_{T,P}$. Note, if $\ell \geq 1/2(|I| - 1)$ we do not add any shortcut edge for the corresponding certified box. Next remove all the diagonal edges (D-steps) of cost 0 or 1 from graph $G'_{T,P}$ and obtain graph $G''_{T,P}$.

▶ **Proposition 8.** *If τ is a path from $(t-m, 0)$ to (t, m) in $G_{T,P}$ which is (k, ζ)-approximated by a subset of certified boxes σ by the covering algorithm then there is a path from $(t-m, 0)$ to (t, m) in $G''_{T,P}$ of cost at most $5 \cdot (k \cdot cost_{G_{T,P}}(\tau) + \zeta)$ consisting of shortcut edges corresponding to σ and H and V steps.*

We provide the proof of proposition 8 in the full version. By Lemma 6 and Proposition 8, for t where $w_2 | t$, the cost of a shortest path from $(t - m, 0)$ to (t, m) in $G''_{T,P}$ is bounded by $O(k_t)$. At the same time, any path in $G''_{T,P}$ from $(i, 0)$ to (t, m), $i \leq t$, has cost at least k_t. So we only need to find the minimal cost of a shortest path from any $(i, 0)$ to (t, m) in $G''_{T,P}$ to get an approximation of k_t.

To find the minimal cost, we reset to zero the cost of all horizontal edges $(i, 0) \to (i + 1, 0)$ in $G''_{T,P}$ to get a graph G. The graph G corresponds to taking the pattern matching graph $\tilde{G}_{T,P}$, removing from it all its diagonal edges and adding the shortcut edges. The cost of a path from $(0, 0)$ to (t, m) in G is the minimum over $i \leq t$ of the cost of a shortest path from $(i, 0)$ to (t, m) in $G''_{T,P}$.

Hence, we want to calculate the cost of the shortest path from $(0, 0)$ to (t, m) for all t.[1] For this we will use a simple algorithm that will make a single sweep over the shortcut edges sorted by their origin and calculate the distances for $t = 0, \ldots, n$. The algorithm will maintain a data structure that at time t will allow to answer efficiently queries about the cost of the shortest path from $(0, 0)$ to (t, j) for any $j \in \{0, \ldots, m\}$.

The data structure will consist of a binary tree with $m + 1$ leaves. Each node is associated with a subinterval of $\{0, \ldots, m\}$ so that the j-th leaf (counting from left to right) corresponds to $\{j\}$, and each internal node corresponds to the union of all its children. We denote by

[1] Although, we really care only about t where $w_2 | t$, as for all the other values of t we will approximate k_t by the value equal to $\tilde{k}_{t'} + (t - t')$, where $t' = \lfloor \frac{t}{w_2} \rfloor$. Recall, $\tilde{k}_{t'}$ is the estimated value of $k_{t'}$.

I_v the interval associated with a node v. The depth of the tree is at most $1 + \log(m + 1)$. At time t, query to the node v of the data structure will return the cost of the shortest path from $(0, 0)$ to $(t, \max I_v)$ that uses some shortcut edge $(i, j) \to (i', j')$, where $j' \in I_v$. Each node v of the data structure stores a pair of numbers (c_v, t_v), where c_v is the cost of the relevant shortest path from $(0, 0)$ to $(t_v, \max I_v)$ and t_v is the time it was updated the last time. (Initially this is set to $(\infty, 0)$.) At time $t \geq t_v$, the query to the node v returns $c_v + (t - t_v)$.

At time t to find the cost of the shortest path from $(0, 0)$ to (t, j) we traverse the data structure from the root to the leaf j. Let $v_1, \ldots, v_\ell$ be the left children of the nodes along the path in which we continue to the right child. We query nodes $v_1, \ldots, v_\ell$ to get answers $a_1, \ldots, a_\ell$. The cost of the shortest paths from $(0, 0)$ to (t, j) is $a = \min\{j, a_1 + (j - \max I_{v_1}), a_2 + (j - \max I_{v_2}), \ldots, a_\ell + (j - \max I_{v_\ell})\}$. As each query takes $O(1)$ time to answer, computing the shortest path to (t, j) takes $O(\log m)$ time.

The algorithm that outputs the cheapest cost of any path from $(0, 0)$ to (t, m) in G will process the shortcut edges $(i, j) \to (i', j')$ one by one in the order of increasing i. The algorithm will maintain lists $L_0, \ldots, L_n$ of updates to the data structure to be made before time t. At time t the algorithm first outputs the cost of the shortest path from $(0, 0)$ to (t, m). Then it takes each shortcut edge $(t, j) \to (t', j')$ one by one, $t < t'$. (The algorithm ignores shortcut edges where $t = t'$.) Using the current state of the data structure it calculates the cost c of a shortest path from $(0, 0)$ to (t, j) and adds $(c + d, j')$ to list $L_{t'}$, where d is the cost of the shortcut edge $(t, j) \to (t', j')$.

After processing all edges starting at $(t, \cdot)$ the algorithm performs updates to the data structure according to the list L_{t+1}. Update (c, j) consists of traversing the tree from the root to the leaf j and in each node v updating its current values (c_v, t_v) to the new values $(c'_v, t + 1)$, where $c'_v = \min\{c_v + t + 1 - t_v, c + \max I_v - j\}$. Then the algorithm increments t and continues with further edges.

If the number of shortcut edges is m then the algorithm runs in time $O(n + m(\log m + \log m))$. First, it has to set-up the data structure, sort the edges by their origin and then it processes each edge. Processing each edge will require $O(\log m)$ time to find the min-cost path to the originating vertex and then later at time t' it will require time $O(\log m)$ to update the data structure. As there are $\widetilde{O}(\frac{n}{m} \cdot \frac{m}{\theta w_1} \cdot \frac{m}{w_1}) \leq \widetilde{O}(nm^{3/4})$ certified boxes in total the running time of the algorithm is as required.

The correctness of the algorithm is immediate from its description.

5 Online approximate pattern matching

In this section we describe the online algorithm from Theorem 2. In the online setting the pattern P is given while the text T arrives in online fashion. The main challenge of this setting is that at any point of time (other than the pattern) we are allowed to store a substring of the text of length just sublinear in m. To overcome this situation the online algorithm is based on interleaved execution of the cover and min-cost path algorithms from Sections 3.2 and 4. Moreover we need to maintain some extra data structure in a clever manner for the covering algorithm. Also to get the required space bound we use a slightly modified tree data structure for the min-cost path algorithm. For the online setting we use the same parameters as the offline one, but we set their values slightly differently: $w_1 = m^{11/18}$, $w_2 = m^{20/27}$, $d = m^{7/54}$, $\theta = m^{-1/9}$. Next, we describe the data structure used in the covering algorithm and the modified tree data structure for the min-cost path algorithm.

Covering algorithm data structure. For each substring T_r^0 of m consecutive input symbols of text T, and $j = \lceil \log 1/\theta \rceil, \ldots, 0$ the algorithm will maintain a set D'_j that stores the content of strings T_i^1 that reached Step 3 of the covering algorithm during processing of T_r^0. Moreover for each of such string T_i^1 the algorithm will also store a set $Y_{i,j}$ that contains the spans obtained in Step 4 while processing T_i^1. This is done as the whole m length string T_r^0 can't be stored at once. Moreover to bound the size of D'_j and $Y_{i,j}$, before adding a new T_i^1 that reached Step 3 of the covering algorithm to D'_j, we first ensure that no string close to T_i^1 is already contained in D'_j. Also after finishing each T_r^0 we discard all the information associated with it.

Modified tree data structure. Here we describe the modified tree data structure used for the min-cost path algorithm. Notice, every shortcut edge corresponds to some certified box. Our covering algorithm has $\log 1/\theta$ rounds where in any round the total number of possible vertical positions, where the bottom left corner or the top right corner point of any certified box might lie is bounded by $\frac{m}{\theta w_1}$. Next, we round up all the edit distance estimates to powers of two, hence in any certified box there are at most $2 \log m$ positions from which a shortcut edge might start or end. Therefore, the number of distinct vertical positions where these shortcut edges might originate from or lead to is bounded by $q = \frac{2m}{\theta w_1} \cdot \log 1/\theta \cdot \log m$. Thus the tree data structure of the min-cost path algorithm will ever perform updates to at most $q \log m$ distinct nodes. We do not need to store the nodes that are never updated, so the tree data structure will occupy only space $\widetilde{O}(\frac{m}{\theta w_1})$.

5.1 The online algorithm

Now we explain how to interleave the two phases to achieve required time and space bound. The algorithm processes the input text T in batches of w_2 symbols. Upon receipt of the t-th symbol we buffer the symbol, if t is not divisible by w_2 then the algorithm outputs the estimated value for $(t-1)$-th position plus one, i.e., $\tilde{k}_{t-1} + 1$ as the current value k_t and waits for the next symbol. Otherwise we received batch T_ℓ^2 of next w_2 symbols, for $\ell = t/w_2$, and we will proceed as follows.

The covering algorithm in the online setting is similar to the covering algorithm offline setting. However here, we will execute the covering algorithm twice on each T_ℓ^2 where during the first execution the only thing that we will send to the min-cost path algorithm are the certified boxes produced at Step 9, all other modifications to data structures will be discarded. During the second run of the algorithm on T_ℓ^2, we will preserve all modifications to D'_j's and other data structures except we will discard the certified boxes produced at Step 9 (we will not send them to the min-cost path algorithm as they are already sent in the first pass).

Covering algorithm. We now describe how the covering algorithm executes on each T_ℓ^2. The algorithm maintain sets S_j, $j = \lceil \log 1/\theta \rceil, \ldots, 0$ that are empty at the beginning. We partition T_ℓ^2 into $T_g^1, \ldots, T_h^1$ of length w_1, where $g = (\ell - 1) \cdot \frac{w_2}{w_1} + 1$ and $h = g + \frac{w_2}{w_1} - 1$. For $i = g, \ldots, h$ we do the following. For each $j = \lceil \log 1/\theta \rceil, \ldots, 0$, set $\epsilon_j = 2^{-j}$. Check, whether T_j^1 is at edit distance at most $2\epsilon_j w_1$ from some string $T_{i'}^1$ in D'_j. If it is then send the set of all the certified boxes $(I, J, 8\epsilon_j w_1)$ to the min-cost path algorithm, where I is the span of T_i^1 in T and $J \in Y_{i',j}$. If it is not close to any string in D'_j then sample the relevant substring in P as in Step 2 and see how many of them are at edit distance $\leq \epsilon_j w_1$ from T_i^1. If at most $\frac{1}{2} \cdot c_0 \cdot \log n$ of the samples have their edit distance from T_i^1 below $\epsilon_j w_1$ then put index i into S_j and continue for another j and then the next i. Otherwise we execute Step 4 of the algorithm to find set Y. (We always skip Step 3.) We put T_i^1 into D_j and set $Y_{i,j}$ to

Y. During the first execution of the covering algorithm, upon processing all j and i we will directly proceed to the sparse extension sampling part whereas after the second execution of the covering algorithm, we send all the certified boxes $(I, J, 8\epsilon_j w_1)$ to the min-cost path algorithm, where I is the span of T^1_i and $J \in Y_{i,j}$.

In the extension sampling part for each $j = \lceil \log 1/\theta \rceil, \ldots, 0$, we sample from the set S_j the strings T^1_ℓ in Step 7, and we proceed for them as in Steps 8–9. During the first execution of the covering algorithm, for each certified box (I, J, ℓ') produced in Step 9 round up ℓ' to the nearest larger or equal power of two and send the box to the min-cost path algorithm.

Min-cost path algorithm. The min-cost path algorithm receives certified boxes from the covering algorithm and it converts them into corresponding shortcut edges. The algorithm receives the certified boxes at two distinct phases.

Shortcut edges generated after the first execution of the covering algorithm correspond to boxes that were produced at Step 9. These edges are sorted by their originating vertex, stored, and processed at appropriate time steps during the next phase.

During the next phase the algorithm receives boxes $(I, J, 8\epsilon_j w_1)$, where I is the span of some T^1_i and $J \in Y_{i,j}$. It converts them into edges and upon receiving all the edges for a particular T^1_i, it sorts them according to their originating vertex. Then the min-cost path algorithm proceeds from time steps $(i-1) \cdot w_1$ to $i \cdot w_1 - 1$, and processes all stored shortcut edges that originate in these time steps. During these time steps it also updates its tree data structure as in the offline case. Again we use lists for storing pending updates. At any moment of time, the number of unprocessed edges and updates is bounded by the number of edges produced in Step 9 and edges produced for a particular string T^1_i. This is at most $\widetilde{O}(\frac{m}{\theta w_1})$. We conclude by the following lemma:

▶ **Lemma 9.** *Let n and m be large enough integers. Let P be the pattern of length m, T be the text of length n (arriving online one symbol at a time), $1/m \le \theta \le 1$ be a real. Let $\theta w_1 \ge 1$, $w_1 \le \theta w_2$, $w_1 | w_2$ and $w_2 | n$. With probability at least $1 - 1/poly(n)$ the online algorithm for pattern matching runs in amortized time $\widetilde{O}(\frac{m}{d} + \frac{m w_1}{w_2} + d w_2 + \frac{m}{w_1})$ per symbol and in extra space $\widetilde{O}(w_2 + \frac{m}{d\theta} + \frac{m}{w_1 \theta} + \frac{m^2}{\theta^2 w_1^2 d} + d)$.*

We defer the proof of the above lemma to the full version. We instantiate the above lemma for the parameters: $w_1 = m^{11/18}$, $w_2 = m^{20/27}$, $d = m^{7/54}$, $\theta = m^{-1/9}$, to get the following:

▶ **Theorem 10** (Restatement of Theorem 2). *There is a constant $c \ge 1$ so that there is a randomized online algorithm that computes $(c, m^{8/9})$-approximation to approximate pattern matching in amortized time $\tilde{O}(m^{1-(7/54)})$ and extra space $\tilde{O}(m^{1-(1/54)})$ with probability at least $1 - 1/poly(m)$.*

6 Discussion

For our online pattern matching algorithm it can be noticed that there is a clear trade off among the running time, the extra space used by the algorithm and the additive part of the approximation factor. Keeping the running time fixed, decreasing the additive part of the approximation factor (by changing the value of parameter θ) would increase the extra space used, and also keeping the additive error part fixed, decreasing the running time would increase the extra space used.

Open Problem. The online algorithm presented in this paper has non-trivial time and space complexity only for the case when the edit distance between the pattern and the text is high. Therefore, it will be nice to extend our online approximation algorithm for the full range of edit distance, which will be interesting from both theoretical and practical perspectives.

References

1 Amir Abboud and Arturs Backurs. Towards Hardness of Approximation for Polynomial Time Problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 11:1–11:26, 2017.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results for LCS and Other Sequence Similarity Measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.

3 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388, 2016.

4 Amir Abboud and Aviad Rubinstein. Fast and Deterministic Constant Factor Approximation Algorithms for LCS Imply New Circuit Lower Bounds. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 35:1–35:14, 2018.

5 Karl Abrahamson. Generalized String Matching. *SIAM J. Comput.*, 16(6):1039–1051, December 1987.

6 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 377–386, 2010.

7 V. L. Arlazarov, E. A. Dinic, M. A. Konrod, and L. A. Faradzev. On economic construction of the transitive closure of a directed graph. *Dokl. Akad*, Nauk SSSR 194:487–488, 1970. [in Russian]. English translation: Soviet. Math. Dokl. 11 No. 5 (1970), 1209–1210.

8 Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM.

9 Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015.

10 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating Edit Distance within Constant Factor in Truly Sub-Quadratic Time. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 979–990, 2018.

11 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating Edit Distance Within Constant Factor in Truly Sub-Quadratic Time. *CoRR*, abs/1810.03664, 2018. arXiv:1810.03664.

12 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A Black Box for Online Approximate Pattern Matching. In *Combinatorial Pattern Matching, 19th Annual Symposium, CPM 2008, Pisa, Italy, June 18-20, 2008, Proceedings*, pages 143–151, 2008.

13 Raphaël Clifford, Markus Jalsenius, and Benjamin Sach. Cell-probe bounds for online edit distance and other pattern matching problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 552–561, 2015.

14 Raphaël Clifford and Benjamin Sach. Online Approximate Matching with Non-local Distances. In *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009, Lille, France, June 22-24, 2009, Proceedings*, pages 142–153, 2009.

15 Raphaël Clifford and Benjamin Sach. Pseudo-realtime Pattern Matching: Closing the Gap. In *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, pages 101–111, 2010.

16 Richard Cole and Ramesh Hariharan. Approximate String Matching: A Simpler Faster Algorithm. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California, USA.*, pages 463–472, 1998.

17 Maxime Crochemore. String-Matching on Ordered Alphabets. *Theor. Comput. Sci.*, 92(1):33–47, 1992.

18 Maxime Crochemore, Leszek Gasieniec, Wojciech Plandowski, and Wojciech Rytter. Two-Dimensional Pattern Matching in Linear Time and Small Space. In *STACS*, pages 181–192, 1995.

19 Zvi Galil and Raffaele Giancarlo. Data structures and algorithms for approximate string matching. *J. Complexity*, 4(1):33–72, 1988.

20 Zvi Galil and Kunsoo Park. An Improved Algorithm for Approximate String Matching. *SIAM Journal on Computing*, 19(6):989–999, 1990.

21 Zvi Galil and Joel Seiferas. Time-space-optimal String Matching (Preliminary Report). In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 106–113, New York, NY, USA, 1981. ACM.

22 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. pBWT: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 397–407, 2017.

23 Leszek Gasieniec, Wojciech Plandowski, and Wojciech Rytter. The Zooming Method: A Recursive Approach to Time-Space Efficient String-Matching. *Theor. Comput. Sci.*, 147(1&2):19–30, 1995.

24 Piotr Indyk. Faster Algorithms for String Matching Problems: Matching the Convolution Bound. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 166–173, 1998.

25 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast Pattern Matching in Strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

26 Tsvi Kopelowitz and Ely Porat. A Simple Algorithm for Approximating the Text-To-Pattern Hamming Distance. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 10:1–10:5, 2018.

27 Gad M. Landau and Uzi Vishkin. Fast Parallel and Serial Approximate String Matching. *Journal of Algorithms*, 10(2):157–169, 1989.

28 VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

29 William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

30 G. Myers. Incremental alignment algorithms and their applications. *Technical Report*, 1986.

31 Gonzalo Navarro. A Guided Tour to Approximate String Matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

32 Mihai Patrascu. Succincter. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 305–313, 2008.

33 Peter H. Sellers. The Theory and Computation of Evolutionary Distances: pattern recognition. *Journal of Algorithms*, pages 1:359–373, 1980.

34 Tatiana A. Starikovskaya. Communication and Streaming Complexity of Approximate Pattern Matching. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, pages 13:1–13:11, 2017.

35 Esko Ukkonen. Algorithms for Approximate String Matching. *Inf. Control*, 64(1-3):100–118, March 1985.

36 Esko Ukkonen and Derick Wood. Approximate String Matching with Suffix Automata. *Algorithmica*, 10(5):353–364, 1993.

Constructing Faithful Homomorphisms over Fields of Finite Characteristic

Prerona Chatterjee
Tata Institute of Fundamental Research, Mumbai, India
prerona.chatterjee@tifr.res.in

Ramprasad Saptharishi
Tata Institute of Fundamental Research, Mumbai, India
ramprasad@tifr.res.in

Abstract

We study the question of algebraic rank or transcendence degree preserving homomorphisms over finite fields. This concept was first introduced by Beecken et al. [3] and exploited by them and Agrawal et al. [2] to design algebraic independence based identity tests using the Jacobian criterion over characteristic zero fields. An analogue of such constructions over finite characteristic fields were unknown due to the failure of the Jacobian criterion over finite characteristic fields.

Building on a recent criterion of Pandey, Saxena and Sinhababu [14], we construct explicit faithful maps for some natural classes of polynomials in fields of positive characteristic, when a certain parameter called the *inseparable degree* of the underlying polynomials is bounded (this parameter is always 1 in fields of characteristic zero). This presents the first generalisation of some of the results of Beecken, Mittmann and Saxena [3] and Agrawal, Saha, Saptharishi, Saxena [2] in the positive characteristic setting.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Computing methodologies → Symbolic and algebraic manipulation

Keywords and phrases Faithful Homomorphisms, Identity Testing, Algebraic Independence, Finite characteristic fields

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.11

Related Version `https://eccc.weizmann.ac.il/report/2018/212/`

Funding *Prerona Chatterjee*: Supported by a fellowship of the DAE.
Ramprasad Saptharishi: Research supported by Ramanujan Fellowship of DST.

1 Introduction

Multivariate polynomials are fundamental objects in mathematics. These are the primary objects of study in algebraic complexity with regard to classifying their hardness as well as algorithmic tasks involving them. The standard computational model for computing multivariate polynomials is *algebraic circuits*. They are directed acyclic graphs with internal nodes labelled by "+" and "×" gates having the obvious operational semantics, and leaves are labelled by the input variables or field constants.

An important concept about relationships between polynomials is the notion of *algebraic dependence*. A set of polynomials $\mathbf{f} = \{f_1, \ldots, f_m\} \subset \mathbb{F}[\mathbf{x}]$ is said to be *algebraically dependent* if there is some nonzero polynomial combination of them that is zero. Such a nonzero polynomial $A(z_1, \ldots, z_m) \in \mathbb{F}[\mathbf{z}]$, if one exists, for which $A(f_1, \ldots, f_m) = 0$ is called the *annihilating polynomial* for the set $\{f_1, \ldots, f_m\}$. For instance, if $f_1 = x$, $f_2 = y$ and $f_3 = x^2 + y^2$, then $A = z_1^2 + z_2^2 - z_3$ is an annihilator. Note that the underlying field is very important. For example, the polynomials $x + y$ and $x^p + y^p$ are algebraically dependent over $\mathbb{F}_p$, but algebraically independent over a characteristic zero field.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 11; pp. 11:1–11:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Algebraic independence is very well-studied and it is known that algebraically independent subsets of a given set of polynomials form a *matroid* ([13]). Hence, the size of the maximum algebraically independent subset of $\mathbf{f}$ is well-defined and is called the *algebraic rank* or *transcendence degree* of $\mathbf{f}$. We denote it by $\mathsf{algrank}(\mathbf{f}) = \mathsf{algrank}(f_1, \ldots, f_m)$.

Several computational questions arise from the above definition. For instance, given a set of polynomials $\mathbf{f} = \{f_1, \ldots, f_m\}$, each f_i given in its dense representation, can we compute the algebraic rank of this set efficiently? What if the f_i's are provided as algebraic circuits?

Furthermore, in instances when $\mathsf{algrank}(\mathbf{f}) = m - 1$, the smallest degree annihilating polynomial is unique ([9]). There could be various questions about the minimal degree annihilator in this case. For instance, can we compute it efficiently? Kayal [9] showed that even checking if the constant term of the annihilator is zero is NP-hard, and evaluating the annihilator at a given point is #P-hard. In fact, recently Guo, Saxena, Sinhababu [7] showed that even in the general case, checking if the constant term of every annihilator is zero is NP-hard. This effectively means that computing the algebraic rank via properties of the annihilating polynomials would be hard.

Despite this, over fields of characteristic zero, algebraic rank has an alternate characterisation via the Jacobian criterion. Jacobi [8] showed that the algebraic rank of a set of polynomials $\mathbf{f}(\subseteq \mathbb{F}[\mathbf{x}])$ is given by the linear rank (over the rational function field $\mathbb{F}(\mathbf{x})$) of the Jacobian of these polynomials. This immediately yields a randomized polynomial time algorithm to compute the algebraic rank of a given set of polynomials by computing the rank of the Jacobian matrix evaluated at a random point [12, 15, 16, 5].

Faithful homomorphisms and PIT

Algebraic independence shares a lot of similarities with linear independence due to the matroid structure. One natural task is to find a *rank-preserving transformation* in this setting. This is defined by what are called *faithful homomorphisms.*

▶ **Definition 1.1** (Faithful homomorphisms [3]). *Let $\mathbf{f} = \{f_1, \ldots, f_m\} \subseteq \mathbb{F}[\mathbf{x}]$ be a set of polynomials. If $\mathbb{K}$ is an extension field of $\mathbb{F}$, a homomorphism $\Phi : \mathbb{F}[\mathbf{x}] \to \mathbb{K}[\mathbf{y}]$ is said to be an* $\mathbb{F}$-faithful *homomorphism for $\{f_1, \ldots, f_m\}$ if*

$$\mathsf{algrank}_{\mathbb{F}}\{f_1, \ldots, f_m\} = \mathsf{algrank}_{\mathbb{F}}\{\Phi(f_1), \ldots, \Phi(f_m)\}.$$

Ideally, we would like a faithful homomorphism with $|\mathbf{y}| \approx \mathsf{algrank}\{\mathbf{f}\}$ and $\mathbb{K} = \mathbb{F}$. Beecken, Mittmann and Saxena [3] showed that a *generic* $\mathbb{F}$-linear homomorphism to $\mathsf{algrank}(\mathbf{f})$ many variables would be an $\mathbb{F}$-faithful homomorphism with high probability.

One important consequence of faithful homomorphisms is that they preserve nonzeroness of any polynomial composition of $f_1, \ldots, f_m$.

▶ **Lemma 1.2** ([3, 2]). *Suppose $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ and Φ is an $\mathbb{F}$-faithful homomorphism for $\{f_1, \ldots, f_m\}$. Then, for any circuit $C(z_1, \ldots, z_m) \in \mathbb{F}[z_1, \ldots, z_m]$, we have* $C(f_1, \ldots, f_m) = 0 \Leftrightarrow C(\Phi(f_1), \ldots, \Phi(f_m)) = 0$.

Thus, constructing explicit faithful homomorphisms can also be used for polynomial identity testing (PIT), which is the task of checking if a given algebraic circuit C computes the identically zero polynomial. For PIT, the goal is to design a deterministic algorithm that runs in time polynomial in the size of the circuit. There are two types of PIT algorithms, *whitebox* and *blackbox* – in the blackbox setting, we are only provided evaluation access to the circuit and some of its parameters (such as degree, number of variables, size etc.). Thus blackbox PIT algorithms for a class $\mathcal{C}$ is equivalent to constructing a *hitting set*, which is a small list of points in $S \subset \mathbb{F}^n$ such that any nonzero polynomial $f \in \mathcal{C}$ is guaranteed to evaluate to a nonzero value on some $\mathbf{a} \in S$.

It follows from Lemma 1.2 that if we can construct explicit $\mathbb{F}$-faithful homomorphisms for a set $\{f_1, \ldots, f_m\}$ whose algebraic rank is $k \ll n$, then we have a *variable reduction* that preserves the nonzeroness of any composition $C(f_1, \ldots, f_m)$. This approach was used by Beecken, Mittmann and Saxena [3] and Agrawal, Saha, Saptharishi, Saxena [2], in the characteristic zero setting, to design identity tests for several subclasses by constructing faithful maps for $\{f_1, \ldots, f_m\}$ with algebraic rank at most $k = O(1)$, when

- each f_i is a sparse polynomial,
- each f_i is a product of multilinear, variable disjoint, sparse polynomials,
- each f_i is a product of linear polynomials,

and further generalisations.

All the above constructions crucially depend on the fact that the rank of the Jacobian captures algebraic independence. However, this fact is true only over fields of characteristic zero and hence the above results are not known to hold over fields of positive characteristic.

Algebraic independence over finite characteristic

A standard example to exhibit the failure of the Jacobian criterion over fields of finite characteristic, is $\left\{x^{p-1}y, y^{p-1}x\right\}$ – these polynomials are algebraically independent over $\mathbb{F}_p$ but the Jacobian is *not* full-rank over $\mathbb{F}_p$. Pandey, Saxena and Sinhababu [14] characterised the extent of failure of the Jacobian criterion for $\{f_1, \ldots, f_m\}$ by a notion called the *inseparable degree* associated with this set (formally defined in the full version [4]). Over characteristic zero, this is always 1 but over characteristic p this is a power of p. In their work, Pandey et al. presented a Jacobian-like criterion to capture algebraic independence. Informally, each row of the *generalized Jacobian matrix* is obtained by taking the Taylor expansion of $f_i(\mathbf{x} + \mathbf{z})$ about a generic point, and truncating to just the terms of degree up to the *inseparable degree*[1] (formally defined in the full version [4]). The exact characterisation is more involved and is presented in Subsection 2.2 but we just state their theorem here.

▶ **Theorem 1.3.** [14] *Let $\{f_1, \ldots, f_k\}$ be a set of n-variate polynomials over a field $\mathbb{F}$ with inseparable degree t. Then, they are algebraically dependent if and only if*

$$\exists(\alpha_1, \ldots, \alpha_k)(\neq \mathbf{0}) \in \mathbb{F}(\mathbf{z})^k \ \textit{s.t.} \ \sum_{i=1}^{k} \alpha_i \cdot \boldsymbol{\mathcal{H}}_t(f_i) = 0 \mod \langle \boldsymbol{\mathcal{H}}_t(f_1), \ldots, \boldsymbol{\mathcal{H}}_t(f_k) \rangle_{\mathbb{F}(\mathbf{z})}^{\geq 2} + \langle \mathbf{x} \rangle^{t+1}.$$

Although the above statement seems slightly different from the one in [14], it is not too hard to see that they are actually equivalent. In their paper, Pandey et al. have stated their criterion in terms of functional dependence. However, stated this way, it clearly generalises the traditional Jacobian criterion.

In the setting when the *inseparable degree* is constant, this characterisation yields a randomized polynomial time algorithm to compute the algebraic rank. Thus, a natural question is whether this criterion can be used to construct faithful homomorphisms for similar classes of polynomials as studied by Beecken et al. [3] and Agrawal et al. [2].

▶ Remark 1.4. Recently, Guo et al. [7] showed that the task of testing algebraic independence is in $\mathsf{AM} \cap \mathsf{coAM}$ via a very different approach. However, it is unclear if their algorithm also yields constructions of faithful homomorphisms or applications to PIT in restricted settings.

[1] Over characteristic zero, the inseparable degree is 1 and this is just the vector of first order partial derivatives.

1.1 Our Results

Following up on the criterion of Pandey, Saxena and Sinhababu [14] for algebraic independence over finite characteristic, we extend the results of Beecken et al. [3] and Agrawal et al. [2] to construct faithful homomorphisms for some restricted settings. We note that we have not formally defined the term *inseparable degree* yet. Although the definition would be required to precisely understand the criterion of Pandey, Saxena and Sinhababu [14], it is not essential for the proofs in this paper. The interested reader may find these field theoretic preliminaries and formal definitions in the full version of the paper [4].

▶ **Theorem 1.5.** *Let $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ such that* $\mathsf{algrank}\{f_1, \ldots, f_m\} = k$ *and the inseparable degree is t. If t and k are bounded by a constant, then we can construct a polynomial (in the input length) sized list of homomorphisms of the form $\Phi : \mathbb{F}[\mathbf{x}] \to \mathbb{F}(s)[y_0, y_1, \ldots, y_k]$ such that at least one of them is guaranteed to be to $\mathbb{F}$-faithful for the set $\{f_1, \ldots, f_m\}$, in the following two settings:*

- *When each of the f_i's are sparse polynomials,*
- *When each of the f_i's are products of variable disjoint, multilinear, sparse polynomials.*

Prior to this, construction of faithful homomorphisms over finite fields was known only in the setting when each f_i has small individual degree [3]. Over characteristic zero fields, the inseparable degree is always 1 and hence the faithful maps constructed in [3], [2] over such fields can be viewed as special cases of our constructions.

The above theorem also holds for a few other models studied by Agrawal et al. [2] (for instance, occur-k products of sparse polynomials). We mention the above two models just as an illustration of lifting the recipe for faithful maps from [3, 2] to the finite characteristic setting.

▶ **Corollary 1.6.** *If $\{f_1, \ldots, f_m\} \in \mathbb{F}[x_1, \ldots, x_n]$ is a set of s-sparse polynomials with algebraic rank k and inseparable degree t where $k, t = O(1)$. Then, for the class of polynomials of the form $C(f_1, \ldots, f_m)$ for any polynomial $C(z_1, \ldots, z_m) \in \mathbb{F}[\mathbf{z}]$, there is an explicit hitting set of size $(s \cdot \deg(C))^{O(1)}$.*

▶ **Corollary 1.7.** *Let $\mathcal{C} = \sum_{i=1}^{m} T_i$ be a depth-4 multilinear circuit of size s, where each T_i is a product of variable-disjoint, s-sparse polynomials. Suppose $\{T_1, \ldots, T_m\} \in \mathbb{F}[x_1, \ldots, x_n]$ is a set of polynomials with algebraic rank k and inseparable degree t where $k, t = O(1)$. Then, for the class of polynomials of the form $C(T_1, \ldots, T_m)$ for any polynomial $C(z_1, \ldots, z_m) \in \mathbb{F}[\mathbf{z}]$, there is an explicit hitting set of size $(s \cdot \deg(C))^{O(1)}$.*

Comparison with the PIT of [14]

Pandey et al. [14] also gives a PIT result for circuits of the form $\sum_i (f_{i,1} \cdots f_{i,m})$ where $\mathsf{algrank}\{f_{i,1}, \ldots, f_{i,m}\} \leq k$ for every i and each $f_{i,j}$ is a degree d polynomial in $\mathbb{F}[x_1, \ldots, x_n]$. They extend the result of Kumar and Saraf [11] to arbitrary fields by giving quasi-polynomial time hitting sets if kd is at most poly-logarithmically large.

Corollary 1.7 however is incomparable to the PIT of Pandey et al. [14] for the following reasons:

- The algebraic rank bound in the case of [14, 11] is a gate-wise bound rather than a global bound. Thus, in principle, it could be the case that $\mathsf{algrank}\{f_{i,1}, \ldots, f_{i,m}\}$ is bounded by k for each i but this would not necessarily translate to a bound on $\mathsf{algrank}\left\{\prod_j f_{i,j} \ : \ i\right\}$ as demanded in Corollary 1.7. Hence, in this regard, the PIT of [14, 11] is stronger.

- In the regime when we have $\mathsf{algrank}\left\{\prod_j f_{i,j} \;:\; i\right\}$ and the inseparable degree of this set to be bounded by a constant, Corollary 1.7 presents an explicit hitting set of polynomial size, whereas it is unclear if [14, 11] provide any non-trivial upper bound as this does not translate to any bound on $\mathsf{algrank}\,\{f_{i,1}, \ldots, f_{i,m}\}$.

On other models studied by Agrawal et al. [2]

Our results, in its current form, do not extend directly some of the other models studied by Agrawal et al. [2], most notably larger depth multilinear formulas. The primary hurdle appears to be the *recursive* use of explicit faithful homomorphisms for larger depth formulas. In the characteristic p setting, unfortunately, it is unclear if a bound on the inseparable degree of the original gates can be used to obtain a bound on the inseparable degree of other sets of polynomials considered in the recursive construction of Agrawal et al. [2].

1.2 Proof overview

The general structure of the proof follows the outline of Agrawal et al. [2]'s construction of faithful homomorphisms in the characteristic zero setting. Roughly speaking, this can be described in the following steps:

Step 1 : For a *generic linear map* $\Phi : \mathbf{x} \to \mathbb{F}(s)[y_1, \ldots, y_k]$, write the Jacobian of the set $\{f_1 \circ \Phi, \cdots, f_k \circ \Phi\}$ in terms of the Jacobian of the set $\{f_1, \cdots, f_k\}$. This can be described succinctly as a matrix product of the form

$$J_{\mathbf{y}}(f \circ \Phi) = \Phi(J_{\mathbf{x}}(\mathbf{f})) \cdot J_{\mathbf{y}}(\Phi(\mathbf{x})).$$

Step 2 : We know that $J_{\mathbf{x}}(\mathbf{f})$ is full rank. Ensure that $\Phi(J_{\mathbf{x}}(\mathbf{f}))$ (where Φ is applied to every entry of the matrix $J_{\mathbf{x}}(\mathbf{f})$) remains full rank. This can be done if $\mathbf{f}$'s are some structured polynomials such as sparse polynomials, or variable-disjoint products of sparse polynomials etc.

Step 3 : Choose the map Φ so as to ensure that

$$\operatorname{rank}(\Phi(J_{\mathbf{x}}(\mathbf{f})) \cdot J_{\mathbf{y}}(\Phi(\mathbf{x}))) = \operatorname{rank}(\Phi(J_{\mathbf{x}}(\mathbf{f}))).$$

This is typically achieved by choosing Φ so as to make $J_{\mathbf{y}}(\Phi(\mathbf{x}))$ a *rank-extractor*. It was shown by Gabizon and Raz [6] that a parametrized Vandermonde matrix has this property, and this allows one to work with a homomorphism of the form (loosely speaking)

$$\Phi : x_i \mapsto \sum_{j=1}^{k} s^{ij} y_j.$$

We would like to execute essentially the same sketch over fields of finite characteristic but we encounter some immediate difficulties. The criterion of Pandey et al. [14] over finite characteristic is more involved but it is reasonably straightforward to execute Steps 1 and 2 in the above sketch using the chain rule of (Hasse) derivatives. The primary issue is in executing Step 3 and this is for two very different reasons.

The first is that, unlike in the characteristic zero setting, the analogue of the matrix $J_{\mathbf{y}}(\Phi(\mathbf{x}))$ have many correlated entries. In the characteristic zero setting, we have complete freedom to choose Φ so that $J_{\mathbf{y}}(\Phi(\mathbf{x}))$ can be any matrix that we want. Roughly speaking, we only have $n \cdot k$ parameters to define Φ but the analogue of $J_{\mathbf{y}}(\Phi(\mathbf{x}))$ is much larger in the finite characteristic setting. Fortunately, there is just about enough structure in the matrix that we can show that it continues to have some rank-preserving properties. This is done in Section 3.

The second hurdle comes from the subspace that we need to work with in the modified criterion. The *rank-extractor* is essentially parametrized by the variable s. In order to show that it preserves the rank of $\Phi(J_{\mathbf{x}}(\mathbf{f}))$ under right multiplication, we would like ensure that the variable s effectively does not appear in this matrix. In the characteristic zero setting, this is done by suitable restriction on other variables to remove any dependencies on s in $\Phi(J_{\mathbf{x}}(\mathbf{f}))$. Unfortunately, in the criterion of Pandey et al. [14], we have to work modulo some suitable subspace and these elements introduce other dependencies on s that appear to be hard to remove. Due to this hurdle, we are unable to construct $\mathbb{F}(s)$-faithful homomorphisms even in restricted settings.

However, we observe that for the PIT applications, we are merely required to ensure that $\{f_1 \circ \Phi, \ldots, f_k \circ \Phi\}$ remain $\mathbb{F}$-algebraically independent instead of $\mathbb{F}(s)$-algebraically independent. With this weaker requirement, we can obtain a little more structure in the subspace involved and that lets us effectively execute Step 3.

Structure of the paper

We begin by describing some preliminaries that are necessary to understand the criterion of Pandey, Saxena and Sinhababu [14] in the next section. Following that, in Section 3, we show that certain Vandermonde-like matrices have *rank-preserving properties.* We use these matrices to give a recipe of constructing faithful maps, in Section 4, and execute this for the settings of Theorem 1.5 in Section 5.

2 Preliminaries

2.1 Notations

- For a positive integer m, we will use $[m]$ to denote set $\{1, 2, \ldots, m\}$.
- We will use bold face letters such as $\mathbf{x}$ to denote a set of indexed variables $\{x_1, \ldots, x_n\}$. In most cases the size of this set would be clear from context. Extending this notation, we will use $\mathbf{x}^{\mathbf{e}}$ to denote the monomial $x_1^{e_1} \cdots x_n^{e_n}$.
- For a set of polynomials $f_1, \ldots, f_m$, we will denote by $\langle f_1, \ldots, f_m \rangle_{\mathbb{K}}$ the set of all $\mathbb{K}$-linear combinations of $f_1, \ldots, f_m$. Extending this notation, we will use $\langle f_1, \ldots, f_m \rangle_{\mathbb{K}}^{r}$ to denote the set of all $\mathbb{K}$-linear combinations of r-products $f_{i_1} \cdots f_{i_r}$ (with $i_1, \ldots, i_r \in [m]$) and $\langle f_1, \ldots, f_m \rangle_{\mathbb{K}}^{\geq r}$ similarly. In instances when we just use $\langle f_1, \ldots, f_m \rangle$, we will denote the *ideal* generated by $f_1, \ldots, f_m$.

Hitting set generators

▶ **Definition 2.1** (Hitting set generators (HSG)). *Let $\mathcal{C}$ be a class of n-variate polynomials. A tuple of polynomials $\mathcal{G} = (G_1(\alpha), \ldots, G_n(\alpha))$ is a* hitting set generator *for $\mathcal{C}$ if for every nonzero polynomial $P(\mathbf{x}) \in \mathcal{C}$ we have $P(G_1(\alpha), \ldots, G_n(\alpha))$ is a nonzero polynomial in α.*

The degree of this generator is defined to be $\max \deg(G_i)$*.*

Intuitively, such a tuple can be used to *generate* a hitting set for $\mathcal{C}$ by running over several instantiations of α. Also, it is well known that any hitting set can be transformed into to HSG via interpolation.

Isolating weight assignments

Suppose $\mathsf{wt} : \{x_i\} \to \mathbb{N}$ is a weight assignment for the variables $\{x_1, \ldots, x_n\}$. We can extend it to define the weight of a monomial as follows.

$$\mathsf{wt}(\mathbf{x}^{\mathbf{e}}) = \sum_{i=1}^{n} e_i \cdot \mathsf{wt}(x_i)$$

▶ **Definition 2.2.** *A weight assignment* $\mathsf{wt} : \{x_i\} \to \mathbb{N}$ *is said to be isolating for a set S of monomials if every pair of distinct monomials in S receives distinct weights.*

With this background, we are now ready to state the criterion for algebraic independence over fields of finite characteristic. Similar to the Jacobian Criterion, Pandey, Saxena and Sinhababu [14] reduce the problem of checking algebraic independence to that of checking linear independence. However, their criterion is slightly more subtle in the sense that we will have to check the linear independence of a set of vectors modulo a large subspace.

A formal statement of the Jacobian criterion along with some field theoretic preliminaries are present in the full version [4]. These include the formal definition of terms such as *inseparable degree* etc. to precisely understand the criterion of Pandey, Saxena and Sinhababu [14] but are not essential for the proof in this paper.

2.2 The PSS Criterion over fields of finite characteristic

In this section we present a slightly different perspective on the criterion of Pandey et al. [14]. A more elaborate discussion of their criterion is deferred to the full version [4].

Define the following operator $\mathcal{H}_t(f) := \deg_{\leq t}(f(\mathbf{x}+\mathbf{z}) - f(\mathbf{z}))$, where $\deg_{\leq t}$ restricts to just those monomials in $\mathbf{x}$ of degree at most t. It is also worth noting that $\mathcal{H}_t(f)$ does not have a constant term and this would become useful in the criterion.

The operator $\mathcal{H}_t$ however, as defined above, is indexed by t. Pandey et al. [14] show that the correct value of t to work with is the *inseparable degree* of the given set of polynomials (see full version [4] for details).

Let $\mathcal{U}_t(\mathbf{f}) = \mathcal{U}_t(f_1, \ldots, f_k)$ denote the subspace $\langle \mathcal{H}_t(f_1), \ldots, \mathcal{H}_t(f_k)\rangle^{\geq 2}_{\mathbb{F}(\mathbf{z})} \bmod \langle \mathbf{x}\rangle^{t+1}$. Then, for any $h \in \mathcal{U}_t(\mathbf{f})$, we define the modified Jacobian matrix as follows.

$$\mathsf{PSSJac}_t(\mathbf{f}, h) = \begin{bmatrix} \mathcal{H}_t(f_1) + h \\ \mathcal{H}_t(f_2) \\ \vdots \\ \mathcal{H}_t(f_k) \end{bmatrix}.$$

The columns of this matrix are indexed by monomials in $\mathbf{x}$ and entries in the column indexed by $\mathbf{x}^{\mathbf{e}}$ are the coefficient of $\mathbf{x}^{\mathbf{e}}$ in the corresponding rows.

▶ **Theorem 2.3** (Alternate Statement for the PSS-criterion). *Let $\{f_1, \ldots, f_k\}$ be a set of n-variate polynomials over a field $\mathbb{F}$ with inseparable degree t. Then, they are algebraically independent if and only if for every $h \in \mathcal{U}_t(\mathbf{f})$,* $\mathsf{PSSJac}_t(\mathbf{f}, h)$ *is full rank.*

Let $\mathcal{V}_t(g_1, \ldots, g_k)$ denote the subspace $\langle \mathcal{H}_t(g_1), \ldots, \mathcal{H}_t(g_k)\rangle^{\geq 2}_{\mathbb{F}(\mathbf{g}(\mathbf{v}))} \bmod \langle \mathbf{y}\rangle^{t+1}$. The following lemma can be inferred in the dependent case.

▶ **Lemma 2.4.** *Let $\mathbb{F}$ any field and $\mathbb{K}$ be an extension field of $\mathbb{F}$. If $\{g_1, \ldots, g_k\}$ is a set of n-variate polynomials in $\mathbb{K}[\mathbf{y}]$ that are $\mathbb{F}$-algebraically dependent, then for any positive integer t, there exists $h' \in \mathcal{V}_t(g_1, \ldots, g_k)$ such that* $\mathsf{PSSJac}_t(\mathbf{g}, h')$ *is not full rank.*

A proof is given in the full version [4] for the sake of completeness, but we note that the steps are almost identical to those in [14].

3 Rank Condensers from Isolating Weight Assignments

In this section, we focus on *rank-preserving* properties of certain types of matrices. These are slight generalisations of similar properties of Vandermonde matrices that were proved by Gabizon and Raz [6] that would be necessary for the application to constructing faithful homomorphisms.

▶ **Lemma 3.1.** *Suppose we have an $n \times n$ matrix V given by*

$$V = \left(\left(s^{j \cdot w_i}\right)\right)_{i,j}$$

where $w_i < w_j$ whenever $i < j$. If V' is a matrix obtained from V by replacing some of the non-diagonal entries by zero, then $\det(V') \neq 0$ and furthermore $\deg(\det(V')) = \sum_{i=1}^{n} i \cdot w_i$.

The proof of this lemma is not too hard, and can be found in the full version [4]. The following lemma extends this to *rank-preserving* properties of a related matrix.

▶ **Lemma 3.2.** *Let A be a matrix over a field $\mathbb{F}$ with k rows and columns indexed by monomials in $\mathbf{x}$ of degree at most D that is full-rank. Further, let $w = (w_1, \ldots, w_n)$ be an isolating weight assignment for the set of degree D monomials, and let $\mathsf{wt}(\mathbf{x}^{\mathbf{e}}) = \sum_{i=1}^{n} w_i e_i$.*

Suppose M_Φ is a matrix whose rows are indexed by monomials in $\mathbf{x}$ of degree at most D, and columns indexed by pure monomials $\left\{y_i^d \ : \ i \in \{1, \ldots, k\} \ , \ d \leq D\right\}$ *given by*

$$M_\Phi(\mathbf{x}^{\mathbf{e}}, y_i^d) = \begin{cases} s^{i \cdot \mathsf{wt}(\mathbf{x}^{\mathbf{e}})} & \textit{if } \deg(\mathbf{x}^{\mathbf{e}}) = d \\ 0 & \textit{otherwise} \end{cases} .$$

where s is a formal variable. Then, $\mathrm{rank}_{\mathbb{F}(s)}(A \cdot M_\Phi) = \mathrm{rank}_{\mathbb{F}}(A)$.

Proof. By the Cauchy-Binet formula, if we restrict M'_Φ to a set T of k-columns, then

$$\det(A \cdot M'_\Phi[T]) = \sum_{\substack{S \subseteq \mathrm{Columns}(A) \\ |S| = k}} \det(A[S]) \cdot \det(M'_\Phi[S, T])$$

We wish to show that the above sum is nonzero for some choice of columns T. We do that by first defining a weight function on minors of A, then proving that there is a unique nonzero minor of A of largest weight, and then choosing a set of columns T such that the degree of $\det(M'_\Phi[S, T])$ coincides with this chosen weight function. Define the *weight* of a minor of A as follows:

Suppose the columns of the minor is indexed by $S = \{\mathbf{x}^{\mathbf{e_1}}, \ldots, \mathbf{x}^{\mathbf{e_k}}\}$ with the property that $\mathsf{wt}(\mathbf{x}^{\mathbf{e_1}}) < \mathsf{wt}(\mathbf{x}^{\mathbf{e_2}}) < \cdots < \mathsf{wt}(\mathbf{x}^{\mathbf{e_k}})$. Define the weight of this minor as

$$\mathsf{wt}(S) = \sum_{i=1}^{k} i \cdot \mathsf{wt}(\mathbf{x}^{\mathbf{e_i}})$$

where, recall, $\mathsf{wt}(\mathbf{x}^{\mathbf{e_i}}) = \sum_j w_j \cdot \mathbf{e_i}(j)$.

▷ Claim 3.3. There is a unique nonzero $k \times k$ minor of A of maximum weight.

Proof. Suppose S_1 and S_2 are two different minors of A with the same weight. We will just identify S_1 and S_2 by the set of column indices for simplicity. Say S_1 has columns indexed by $\mathbf{x}^{\mathbf{e_1}}, \ldots, \mathbf{x}^{\mathbf{e_k}}$ with $\mathsf{wt}(\mathbf{x}^{\mathbf{e_1}}) < \mathsf{wt}(\mathbf{x}^{\mathbf{e_2}}) < \cdots < \mathsf{wt}(\mathbf{x}^{\mathbf{e_k}})$ and S_2 has columns indexed by $\mathbf{x}^{\mathbf{e'_1}}, \ldots, \mathbf{x}^{\mathbf{e'_k}}$ with $\mathsf{wt}(\mathbf{x}^{\mathbf{e'_1}}) < \mathsf{wt}(\mathbf{x}^{\mathbf{e'_2}}) < \cdots < \mathsf{wt}(\mathbf{x}^{\mathbf{e'_k}})$.

Suppose S_1 and S_2 agree on the first i columns, that is $\mathbf{e}_j = \mathbf{e}'_j$ for all $j \leq i$, and say $\mathsf{wt}(\mathbf{e}_{i+1}) < \mathsf{wt}(\mathbf{e}'_{i+1})$. By the matroid property, there must be some column $\mathbf{x}^{\mathbf{e}'_j}$ from S_2 that we can add to $S_1 \setminus \{\mathbf{x}^{\mathbf{e}_{i+1}}\}$ so that $S = S_1 \setminus \{\mathbf{x}^{\mathbf{e}_{i+1}}\} \cup \left\{\mathbf{x}^{\mathbf{e}'_j}\right\}$ is also a nonzero minor of A. Suppose that

$$\mathsf{wt}(\mathbf{x}^{\mathbf{e}_1}) < \cdots < \mathsf{wt}(\mathbf{x}^{\mathbf{e}_{i+r}}) < \mathsf{wt}(\mathbf{x}^{\mathbf{e}'_j}) < \mathsf{wt}(\mathbf{x}^{\mathbf{e}_{i+r+1}}) < \cdots < \mathsf{wt}(\mathbf{x}^{\mathbf{e}_k}).$$

Then,

$$\begin{aligned}\mathsf{wt}(S) &= \sum_{a=1}^{i} a \cdot \mathsf{wt}(\mathbf{x}^{\mathbf{e}_a}) + \sum_{a=i+2}^{i+r} (a-1) \cdot \mathsf{wt}(\mathbf{x}^{\mathbf{e}_a}) + (i+r)\,\mathsf{wt}(\mathbf{x}^{\mathbf{e}'_j}) + \sum_{a=i+r+1}^{k} a \cdot \mathsf{wt}(\mathbf{x}^{\mathbf{e}_a})\\ &> \sum_{a=1}^{i} a \cdot \mathsf{wt}(\mathbf{x}^{\mathbf{e}_a}) + (i+1)\,\mathsf{wt}(\mathbf{x}^{\mathbf{e}'_j}) + \sum_{a=i+2}^{k} a \cdot \mathsf{wt}(\mathbf{x}^{\mathbf{e}_a}) > \sum_{a=1}^{k} a \cdot \mathsf{wt}(\mathbf{x}^{\mathbf{e}_a}) = \mathsf{wt}(S_1)\end{aligned}$$

Hence, there cannot be two different nonzero minors of A of the same weight. Thus, the nonzero minor of largest weight is unique. ◁

We will now choose k columns from M'_Φ as follows in such a way that the degree of the corresponding determinant agrees with the weight function. Note that the matrix M'_Φ has a natural block-diagonal structure based on the degree of the monomials indexing the rows and columns.

- Let S_0 be the unique $k \times k$ minor of A having maximum weight. Further, assume its columns are indexed by $\mathbf{x}^{\mathbf{e}_1}, \ldots, \mathbf{x}^{\mathbf{e}_k}$ with $\mathsf{wt}(\mathbf{x}^{\mathbf{e}_1}) < \mathsf{wt}(\mathbf{x}^{\mathbf{e}_2}) < \ldots < \mathsf{wt}(\mathbf{x}^{\mathbf{e}_k})$. Let $d_i = \deg(\mathbf{x}^{\mathbf{e}_i}) = \sum_j (\mathbf{e}_i)_j$.
- Choose the columns $T = \left\{y_1^{d_1}, y_2^{d_2}, \ldots, y_k^{d_k}\right\}$ of the matrix M'_Φ.

By Lemma 3.1, for any set of $S' \subseteq \mathrm{Columns}(A)$, we have $\deg(\det(M_\Phi[S', T])) \leq \mathsf{wt}(S')$ and furthermore we also have $\deg(M'_\Phi[S_0, T]) = \mathsf{wt}(S_0)$ as we chose the columns T to ensure that the main diagonal of the sub-matrix has only nonzero elements. Hence,

$$\det(A \cdot M'_\Phi[T]) = \sum_{\substack{S \subseteq \mathrm{Columns}(A)\\ |S| = k}} \det(A[S]) \cdot \det(M'_\Phi[S, T]) \neq 0$$

since the contribution from $A[S_0] \det(M'_\Phi[S_0, T])$ is the unique term of highest degree and so cannot be cancelled. ◀

4 Construction of Explicit Faithful Maps

We will be interested in applying a map $\Phi : \mathbb{F}[\mathbf{x}] \to \mathbb{F}(s)[\mathbf{y}]$ and study the transformation of the PSS-Jacobian. Since the entries of the PSS-Jacobian involve $\mathcal{H}_t(f(\mathbf{x})) = \deg_{\leq t}(f(\mathbf{x}+\mathbf{z}) - f(\mathbf{z}))$, we would need to also work with $\mathcal{H}_t(g(\mathbf{y}))$ where $g(\mathbf{y}) = f \circ \Phi$. To make it easier to follow, we shall use a different name for the variables in the two cases. Hence,

$$\mathcal{H}_t(f(\mathbf{x})) := \deg_{\leq t}(f(\mathbf{x}+\mathbf{z}) - f(\mathbf{z})) \quad , \quad \mathcal{H}_t(g(\mathbf{y})) := \deg_{\leq t}(g(\mathbf{y}+\mathbf{v}) - g(\mathbf{v}))\,.$$

4.1 Recipe for constructing faithful maps

Let $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ be polynomials with $\mathsf{algrank}\,\{f_1, \ldots, f_m\} = k$ and inseparable degree t. We will work with linear transformations of the form:

$$\Phi : x_i \mapsto a_i y_0 + \sum_{j=1}^{k} s^{w_i \cdot j} y_j, \quad \text{for all } i \in [n],$$

$$\Phi_z : z_i \mapsto a_i v_0 + \sum_{j=1}^{k} s^{w_i \cdot j} v_j, \quad \text{for all } i \in [n].$$

where all the variables on the RHS are formal variables. Further, define $\{g_1, \ldots, g_m\} \in \mathbb{F}[\mathbf{z}]$ as $g_i = f_i \circ \Phi$ and $\boldsymbol{\mathcal{H}}_t(g_i) = \deg_{\leq t}(g_i(\mathbf{y} + \mathbf{v}) - g_i(\mathbf{v}))$.

The main lemma of this section is the following *recipe* for constructing faithful maps.

▶ **Lemma 4.1** (Recipe for faithful homomorphisms). *Let $f_1, \ldots, f_m \in \mathbb{F}[\mathbf{x}]$ be polynomials such that their algebraic rank is at most k and suppose the inseparable degree is bounded by a constant t. Further,*

- *suppose $\mathcal{G} = (G_1(\alpha), \ldots, G_n(\alpha))$ is a* hitting-set generator (HSG) *for the class of all $k \times k$ minors of $\mathsf{PSSJac}_t(\mathbf{f}, h)$ for any $h \in \mathcal{U}_t(\mathbf{f})$.*
- *suppose $w : [n] \to \mathbb{N}$ is an isolating weight assignment for the set of n-variate monomials of degree at most t.*

Then, the homomorphism $\Phi : \mathbb{F}[x_1, \ldots, x_n] \to \mathbb{F}(s, \alpha)[y_0, \ldots, y_k]$ defined as

$$\Phi : x_i \mapsto y_0 G_i(\alpha) + \sum_{j=1}^{k} y_j \cdot s^{w(i)j},$$

is an $\mathbb{F}$-faithful homomorphism for the set $\{f_1, \ldots, f_m\}$.

As mentioned earlier, the rough proof sketch would be to first write the PSS-Jacobian of the transformed polynomials $\mathbf{g}$ in terms of $\mathbf{f}$, express that as a suitable matrix product, and use some *rank extractor* properties of the associated matrix, as described in Section 3. So first, let us see how we can get the required matrix product.

▶ **Lemma 4.2** (Evolution of polynomials under Φ). *Let $\Phi : \mathbf{x} \to \mathbb{F}(s)[\mathbf{y}]$ and $\Phi_z : \mathbf{z} \to \mathbb{F}(s)[\mathbf{v}]$ be given as above. Further, for any polynomial $h'(a_1, \ldots, a_m) \in \mathbb{F}(\mathbf{g}(\mathbf{v}))[\mathbf{a}]$, define $h(a_1, \ldots, a_m) \in \mathbb{F}(\mathbf{f}(\mathbf{z}))[\mathbf{a}]$ as follows.*

$\mathsf{coeff}_{\mathbf{a}^{\mathbf{e}}}(h)$ *is got by replacing every occurrence of $g_i(\mathbf{v})$ by $f_i(\mathbf{z})$ in* $\mathsf{coeff}_{\mathbf{a}^{\mathbf{e}}}(h')$

Then,

$$h'(\boldsymbol{\mathcal{H}}_t(g_1), \ldots, \boldsymbol{\mathcal{H}}_t(g_m)) = \Phi \circ \Phi_z(h(\boldsymbol{\mathcal{H}}_t(f_1), \ldots, \boldsymbol{\mathcal{H}}_t(f_m))).$$

It is worth noting that the polynomial $h(a_1, \ldots, a_m)$ is independent of s. This would be crucial later on in the proof. The proof of this lemma is not too hard and can be found in the full version [4].

▶ **Corollary 4.3** (Matrix representation of the evolution). *Suppose A' is a matrix whose columns are indexed by monomials in $\mathbf{y}$. Further suppose a row in A' corresponds to a polynomial, say $h'(\boldsymbol{\mathcal{H}}_t(g_1), \ldots, \boldsymbol{\mathcal{H}}_t(g_m)) \in \mathbb{F}(\mathbf{g}(\mathbf{v}))[\mathbf{y}]$, whose entry in the column indexed by $\mathbf{y}^{\mathbf{e}}$ is $\mathsf{coeff}_{\mathbf{y}^{\mathbf{e}}}(h'(\boldsymbol{\mathcal{H}}_t(\mathbf{g}))) \in \mathbb{F}(\mathbf{v}, \mathbf{s})$. If A is the corresponding matrix (having entries from $\mathbb{F}(\mathbf{z})$) with columns indexed by monomials in $\mathbf{x}$ and the corresponding row being $h(\boldsymbol{\mathcal{H}}_t(f_1), \ldots, \boldsymbol{\mathcal{H}}_t(f_m)) \in \mathbb{F}(\mathbf{f}(\mathbf{z}))[\mathbf{x}]$ as described in Lemma 4.2, then*

$$A' = \Phi_z(A) \times \widetilde{M_\Phi}$$

where $\widetilde{M_\Phi}(\mathbf{x}^{\mathbf{e}}, \mathbf{y}^{\mathbf{d}}) = \mathsf{coeff}_{\mathbf{y}^{\mathbf{d}}}(\Phi(\mathbf{x}^{\mathbf{e}}))$.

Using these and Lemma 3.2, we are now ready to prove Lemma 4.1.

Proof of Lemma 4.1. Without loss of generality, say $\{f_1, \ldots, f_k\}$ is an algebraically independent set. We wish to show that if $g_i = f_i \circ \Phi$, then $\{g_1, \ldots, g_k\}$ is an $\mathbb{F}$-algebraically independent set as well. Assume on the contrary that $\{g_1, \ldots, g_k\}$ is an $\mathbb{F}$-algebraically dependent set. Then for t being the inseparable degree of $\{f_1, \ldots, f_k\}$, by Lemma 2.4, there exists

$$h' \in \mathcal{V}_t(g_1, \ldots, g_k) := \langle \boldsymbol{\mathcal{H}}_t(g_1), \ldots, \boldsymbol{\mathcal{H}}_t(g_k)\rangle^{\geq 2}_{\mathbb{F}(\mathbf{g}(\mathbf{v}))} \bmod \langle \mathbf{y}\rangle^{t+1}$$

such that $\mathsf{PSSJac}_t(\mathbf{g}, h')$ is not full rank. Without loss of generality, we can assume that the entries of $\mathsf{PSSJac}_t(\mathbf{g}, h')$ are denominator-free by clearing out any denominators. Corresponding to h', define h as in Lemma 4.2, which would also satisfy that

$$h \in \mathcal{U}_t(f_1, \ldots, f_k) := \langle \boldsymbol{\mathcal{H}}_t(f_1), \ldots, \boldsymbol{\mathcal{H}}_t(f_k)\rangle^{\geq 2}_{\mathbb{F}(\mathbf{z})} \bmod \langle \mathbf{x}\rangle^{t+1}.$$

It is worth stressing the fact that the polynomial h is independent of the variable s. Then by Corollary 4.3 we get

$$\mathsf{PSSJac}_t(\mathbf{g}, h') = \Phi_z(\mathsf{PSSJac}_t(\mathbf{f}, h)) \times \widetilde{M_\Phi}.$$

Now, if we substitute $v_0 = 1$ and $v_i = 0$ for every $i \in [k]$, we get

$$\mathsf{PSSJac}_t(\mathbf{g}, h')(v_0 = 1, v_1 = \ldots = v_k = 0) = \mathsf{PSSJac}_t(\mathbf{f}, h)(\mathbf{z} = \mathbf{G}(\alpha)) \times \widetilde{M_\Phi}.$$

But since $\{f_1, \ldots, f_k\}$ is algebraically independent, Theorem 2.3 yields that $\mathsf{PSSJac}_t(\mathbf{f}, h)$ has full rank. Thus, $\mathsf{PSSJac}_t(\mathbf{f}, h)(\mathbf{z} = \mathbf{G}(\alpha))$ also has full rank since $\mathcal{G} = (G_1(\alpha), \ldots, G_n(\alpha))$ is a hitting-set generator for the class of all $k \times k$ minors of $\mathsf{PSSJac}_t(\mathbf{f}, h)$. Most crucially, the matrix $\mathsf{PSSJac}_t(\mathbf{f}, h)$ is independent of the variable s.

To complete the proof, we need to show that multiplication by $\widetilde{M_\Phi}$ continues to keep this full rank to contradict the initial assumption that $\mathsf{PSSJac}_t(\mathbf{g}, h')$ was not full rank.

Finally note that for the Φ we have defined, $\widetilde{M_\Phi}$ restricted to only the *pure monomial* columns

$$\left\{y_i^j \ : \ i \in \{1, \ldots, k\} \ , \ j \in \{0, 1, \ldots, t\}\right\},$$

is the same as M_Φ as defined in Lemma 3.2. Further, w is an isolating weight assignment for the set of n-variate monomials of degree at most t, we satisfy the requirements of Lemma 3.2. Hence, by Lemma 3.2,

$$\begin{aligned} \operatorname{rank}_{\mathbb{F}(s,\alpha)} (\mathsf{PSSJac}_t(\mathbf{g}, h')(v_0 = 1, v_1 = \ldots, v_k = 0)) &= \operatorname{rank}_{\mathbb{F}(\alpha)} \mathsf{PSSJac}_t(\mathbf{f}, h)(\mathbf{z} = \mathbf{G}(\alpha)) \\ \implies \operatorname{rank}_{\mathbb{F}(s,\alpha,\mathbf{v})} (\mathsf{PSSJac}_t(\mathbf{g}, h')) &\geq \operatorname{rank}_{\mathbb{F}(\alpha)} \mathsf{PSSJac}_t(\mathbf{f}, h)(\mathbf{z} = \mathbf{G}(\alpha)) \\ &= k, \end{aligned}$$

which contradicts our assumption that it was not full rank. Hence, it must indeed be the case that $\{f_1 \circ \Phi, \ldots, f_k \circ \Phi\}$ is $\mathbb{F}$ - algebraically independent. ◀

5 Explicit faithful maps and PIT applications in restricted settings

We now describe some specific instantiations of the recipe given by Lemma 4.1 in restricted settings. Let us first recall the statement of the main theorem.

▶ **Theorem 1.5.** *Let $f_1, \dots, f_m \in \mathbb{F}[x_1, \dots, x_n]$ such that* $\mathsf{algrank}\{f_1, \dots, f_m\} = k$ *and the inseparable degree is t. If t and k are bounded by a constant, then we can construct a polynomial (in the input length) sized list of homomorphisms of the form $\Phi : \mathbb{F}[\mathbf{x}] \to \mathbb{F}(s)[y_0, y_1, \dots, y_k]$ such that at least one of them is guaranteed to be to $\mathbb{F}$-faithful for the set $\{f_1, \dots, f_m\}$, in the following two settings:*

- *When each of the f_i's are sparse polynomials,*
- *When each of the f_i's are products of variable disjoint, multilinear, sparse polynomials.*

Proof. By Lemma 4.1, $\Phi : \mathbb{F}[x_1, \dots, x_n] \to \mathbb{F}(s, \alpha)[y_0, \dots, y_k]$ defined as

$$\Phi : x_i \mapsto y_0 G_i(\alpha) + \sum_{j=1}^{k} y_j \cdot s^{w(i)j},$$

is a faithful homomorphism for the set $\{f_1, \dots, f_m\}$ if for any $h \in \mathcal{U}_t(\mathbf{f})$, $w = (w_1, \dots, w_n)$ is a basis isolating weight assignment for $\mathsf{PSSJac}(\mathbf{f}, h)$ and $\mathcal{G} = (G_1(\alpha), \dots, G_n(\alpha))$ is such that the rank of $\mathsf{PSSJac}_t(\mathbf{f}, h)$ is preserved after the substitution $\mathbf{z} \to \mathbf{a}$ for some $\mathbf{a} \in \mathcal{G}$. We define the weight using the standard hashing techniques [10, 1].

Defining w: Define $w : [n] \to \mathbb{N}$ as $w(i) = (t+1)^i \pmod{p}$, where t is the inseparable degree.

Assuming t to be a constant, there are only poly(n) many distinct monomials in $\mathbf{x}$ of degree at most t. Thus, standard results by Klivans and Spielman [10] or Agrawal and Biswas [1] shows that it suffices to go over poly(n) many 'p's before w isolates all monomials in $\mathbf{x}$ of degree at most t.
Let $\mathsf{PSSJac}_t(\mathbf{f})$ be the matrix with columns indexed by monomials in $\mathbf{x}$ of degree at most t and rows by k-variate monomials $\mathbf{a}^{\mathbf{e}}$ in degree at most t, defined as follows.

$$\mathsf{PSSJac}_t(\mathbf{f})[\mathbf{a}^{\mathbf{e}}, \mathbf{x}^{\mathbf{d}}] = \mathsf{coeff}_{\mathbf{x}^{\mathbf{d}}}(\boldsymbol{\mathcal{H}}_t(\mathbf{f})^{\mathbf{e}})$$

Set $K = \binom{k+t}{t}$ be the number of rows in $\mathsf{PSSJac}_t(\mathbf{f})$. Then the following is true.

▷ Claim 5.1. If $\mathcal{G}$ is a hitting set generator for every $K' \times K'$ minor of $\mathsf{PSSJac}_t(\mathbf{f})$ where $K' \leq K$, then the rank of $\mathsf{PSSJac}_t(\mathbf{f}, h)$ is preserved for every $h \in \mathcal{U}_t(\mathbf{f})$.

Proof. We need to show that there is an $\mathbf{a}$ in $\mathcal{G}$ which has the following property:

> For any $h \in \mathcal{U}_t(\mathbf{f})$, if $\{H_t(f_1) + h, H_t(f_2), \dots, H_t(f_k)\}$ are linearly independent, then so are $\{H_t(f_1)(\mathbf{a}) + h(\mathbf{a}), H_t(f_2)(\mathbf{a}), \dots, H_t(f_k)(\mathbf{a})\}$.

Now suppose this is not the case. Then it must be the case that without loss of generality, some $h \in \mathcal{U}_t(\mathbf{f})$, $\mathsf{PSSJac}_t(\mathbf{f}, h)$ has full rank but for any $\mathbf{a} \in \mathcal{G}$,

$$\alpha_1(H_t(f_1)(\mathbf{a}) + h(\mathbf{a})) + \sum_{i=2}^{k} (\alpha_i \cdot H_t(f_i)(\mathbf{a})) = 0.$$

Here, not all of $\{\alpha_i\}_{i \in [k]}$ are zero. However by our hypothesis, this would mean that

$$\alpha_1(H_t(f_1) + h) + \sum_{i=2}^{k} (\alpha_i \cdot H_t(f_i)) \neq 0.$$

Let $\mathcal{B}$ be a basis of the rows in $H_t(\mathbf{f})$. Then each of $\{H_t(f_1)+h, H_t(f_2), \ldots, H_t(f_k)\}$ can be written in terms of rows in $\mathcal{B}$. Thus, the above statement can be rewritten as

$$\sum_{i=1}^{K'} \beta_i \cdot b_i = \alpha_1(H_t(f_1)+h) + \sum_{i=2}^{k}(\alpha_i \cdot H_t(f_i)) \neq 0$$

where $\{\beta_i\}_{i\in[K']}$ are some scalars and $K' = |\mathcal{B}|$.

This shows that not all $\{\beta_i\}_{i=1}^{K'}$ can be zero. Now since $\mathcal{G}$ is a hitting set generator for every $K' \times K'$ minor in $\mathsf{PSSJac}_t(\mathbf{f})$, there is some $\mathbf{a} \in \mathcal{G}$ such that $\{b_i(\mathbf{a})\}_{i\in[K']}$ continue to remain linearly independent. Thus, $\sum_{i=1}^{K'} \beta_i \times b_i(\mathbf{a})! = 0$, since not all $\{\beta_i\}_{i\in[K']}$ is zero. However, this shows that

$$\alpha_1(H_t(f_1)(\mathbf{a}) + h(\mathbf{a})) + \sum_{i=2}^{k}(\alpha_i \cdot H_t(f_i)(\mathbf{a})) = \sum_{i=1}^{K'} \beta_i \times b_i(\mathbf{a}) \neq 0.$$

This contradicts our assumption, and so it must be the case that for any $h \in \mathcal{U}_t(\mathbf{f})$, the rank of $\mathsf{PSSJac}_t(\mathbf{f}, h)$ is preserved. ◁

Thus, now it is only a question of finding a hitting set generator of low degree, for every $K' \times K'$ minor of $\mathsf{PSSJac}_t(\mathbf{f})$ where $K' \leq K$. The definitions of these generators for both cases are similar to those in [2] and the details can be found in the full version [4]. ◀

5.1 Applications to PIT

As stated in Subsection 1.1, using Lemma 1.2, we get two straightforward corollaries for PIT for related models (Corollary 1.6 and Corollary 1.7). As mentioned there, the results are incomparable with the PIT results of Pandey et al. [14] and Kumar and Saraf [11]. For the proof idea, the interested reader may look at the full version [4].

6 Conclusion and open problems

We studied the task of constructing faithful homomorphisms in the finite characteristic setting and extended the results of Agrawal et al. [2] in the setting when the inseparable degree is bounded. There are some very natural open problems in this context.

- Are the homomorphisms constructed in the paper also $\mathbb{F}(s)$-faithful homomorphisms?

  Our proof only provides a recipe towards constructing $\mathbb{F}$-faithful homomorphisms due to technical obstacles involving the criterion for algebraic independence over finite characteristic fields. This is not an issue in characteristic zero fields; Agrawal et al. [2] construct $\mathbb{F}(s)$-faithful homomorphisms.

- How crucial is the notion of inseparable degree in the context of testing algebraic independence?

  The criterion of Pandey, Saxena and Sinhababu [14] crucially depends on this field theoretic notion and there seems to be compelling algebraic reasons to believe that this is necessary. However, as mentioned earlier, Guo, Saxena and Sinhababu [7] showed that algebraic independence testing is in $\mathsf{AM} \cap \mathsf{coAM}$ and this proof has absolutely no dependence on the inseparable degree.

References

1 Manindra Agrawal and Somenath Biswas. Primality and identity testing via Chinese remaindering. *J. ACM*, 50(4):429–443, 2003. doi:10.1145/792538.792540.

2 Manindra Agrawal, Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. Jacobian Hits Circuits: Hitting Sets, Lower Bounds for Depth-D Occur-k Formulas and Depth-3 Transcendence Degree-k Circuits. *SIAM J. Comput.*, 45(4):1533–1562, 2016. doi:10.1137/130910725.

3 Malte Beecken, Johannes Mittmann, and Nitin Saxena. Algebraic independence and blackbox identity testing. *Information and Computing*, 222:2–19, 2013. doi:10.1016/j.ic.2012.10.004.

4 Prerona Chatterjee and Ramprasad Saptharishi. Constructing Faithful Homomorphisms over Fields of Finite Characteristic. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:212, 2018. URL: https://eccc.weizmann.ac.il/report/2018/212.

5 Richard A. DeMillo and Richard J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

6 Ariel Gabizon and Ran Raz. Deterministic extractors for affine sources over large fields. *Combinatorica*, 28(4):415–440, 2008. doi:10.1007/s00493-008-2259-3.

7 Zeyu Guo, Nitin Saxena, and Amit Sinhababu. Algebraic Dependencies and PSPACE Algorithms in Approximative Complexity. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 10:1–10:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CCC.2018.10.

8 C.G.J. Jacobi. De Determinantibus functionalibus. *Journal für die reine und angewandte Mathematik*, 22:319–359, 1841. URL: http://eudml.org/doc/147138.

9 Neeraj Kayal. The Complexity of the Annihilating Polynomial. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 184–193, 2009. doi:10.1109/CCC.2009.37.

10 Adam R. Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 216–223. ACM, 2001. doi:10.1145/380752.380801.

11 Mrinal Kumar and Shubhangi Saraf. Arithmetic Circuits with Locally Low Algebraic Rank. *Theory of Computing*, 13(1):1–33, 2017. doi:10.4086/toc.2017.v013a006.

12 Øystein Ore. Über höhere kongruenzen. *Norsk Mat. Forenings Skrifter*, 1(7):15, 1922.

13 James G. Oxley. *Matroid theory.* Oxford University Press, 1992.

14 Anurag Pandey, Nitin Saxena, and Amit Sinhababu. Algebraic independence over positive characteristic: New criterion and applications to locally low-algebraic-rank circuits. *Computational Complexity*, 27(4):617–670, 2018. doi:10.1007/s00037-018-0167-5.

15 Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

16 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979. doi:10.1007/3-540-09519-5_73.

Maximum-Area Rectangles in a Simple Polygon

Yujin Choi
Technische Universität Berlin, Germany
yj5162@postech.ac.kr

Seungjun Lee
Pohang University of Science and Technology, Pohang, Korea
juny2400@postech.ac.kr

Hee-Kap Ahn
Pohang University of Science and Technology, Pohang, Korea
http://tcs.postech.ac.kr/~heekap
heekap@postech.ac.kr

Abstract

We study the problem of finding maximum-area rectangles contained in a polygon in the plane. There has been a fair amount of work for this problem when the rectangles have to be axis-aligned or when the polygon is convex. We consider this problem in a simple polygon with n vertices, possibly with holes, and with no restriction on the orientation of the rectangles. We present an algorithm that computes a maximum-area rectangle in $O(n^3 \log n)$ time using $O(kn^2)$ space, where k is the number of reflex vertices of P. Our algorithm can report all maximum-area rectangles in the same time using $O(n^3)$ space. We also present a simple algorithm that finds a maximum-area rectangle contained in a convex polygon with n vertices in $O(n^3)$ time using $O(n)$ space.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Maximum-area rectangle, largest rectangle, simple polygon

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.12

Related Version https://arxiv.org/abs/1910.08686

Acknowledgements This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the SW Starlab support program (IITP-2017-0-00905) supervised by the IITP (Institute for Information & communications Technology Promotion).

1 Introduction

Computing a largest figure of a certain prescribed shape contained in a container is a fundamental and important optimization problem in pattern recognition, computer vision and computational geometry. There has been a fair amount of work for finding rectangles of maximum area contained in a convex polygon P with n vertices in the plane. Amenta showed that an axis-aligned rectangle of maximum area can be found in linear time by phrasing it as a convex programming problem [3]. Assuming that the vertices are given in order along the boundary of P, stored in an array or balanced binary search tree in memory, Fischer and Höffgen gave $O(\log^2 n)$-time algorithm for finding an axis-aligned rectangle of maximum area contained in P [9]. The running time was improved to $O(\log n)$ by Alt et al. [2].

Knauer et al. [12] studied a variant of the problem in which a maximum-area rectangle is not restricted to be axis-aligned while it is contained in a convex polygon. They gave randomized and deterministic approximation algorithms for the problem. Recently, Cabello et al. [6] gave an exact $O(n^3)$-time algorithm for finding a maximum-area rectangle with no restriction on its orientation that is contained in a convex n-gon. They also gave an algorithm for finding a maximum-perimeter rectangle and approximation algorithms.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 12; pp. 12:1–12:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This problem has also been studied for containers which are not necessarily convex. Some previous work on the problem focuses on finding an axis-aligned rectangle of maximum area or of maximum perimeter contained in a rectilinear polygonal container in the plane [1, 13, 14]. Daniels et al. studied the problem of finding a maximum-area axis-aligned rectangle contained in a polygon, not necessarily convex and possibly having holes [8]. They gave $O(n \log^2 n)$-time algorithm for the problem. Later, Boland and Urrutia improved the running time by a factor of $O(\log n)$ for simple polygons with n vertices [5]. With no restriction on the orientation of the rectangles, Hall-Holt et al. gave a PTAS for finding a fat[1] rectangle of maximum area contained in a simple polygon [11].

Our results. We study the problem of finding a maximum-area rectangle with no restriction on its orientation that is contained in a simple polygon P with n vertices, possibly with holes, in the plane. We are not aware of any previous work on this problem, except a PTAS for finding a fat rectangle of maximum area inscribed in a simple polygon [11]. We present an algorithm that computes a maximum-area rectangle contained in a simple polygon with n vertices in $O(n^3 \log n)$ time using $O(kn^2)$ space, where k is the number of reflex vertices of P. Our algorithm can also find all rectangles of maximum area contained in P in the same time using (n^3) space. We also present a simple algorithm that finds a maximum-area rectangle contained in a convex polygon with n vertices in $O(n^3)$ time using $O(n)$ space.

To obtain the running time and space complexities, we characterize the maximum-area rectangles and classify them into six types, based on the sets of contacts on their boundaries with the polygon boundary. Then we find a maximum-area rectangle in each type so as to find a maximum-area rectangle contained in P. To facilitate the process, we construct a ray-shooting data structure for P of $O(n)$ space which supports, for a given query point in P and a direction, $O(\log n)$ query time. We also construct the visibility region from each vertex within P, which can be done in $O(n^2)$ time using $O(n^2)$ space in total. For some types, we compute locally maximal rectangles aligned to the coordinate axes while we rotate the axes. To do this, we maintain a few data structures such as double staircases of reflex vertices and priority queues for events during the rotation of the coordinate axes. They can be constructed and maintained in $O(kn^2 \log n)$ time using $O(kn^2)$ space. The total number of events considered by our algorithm is $O(n^3)$, each of which is handled in $O(\log n)$ time.

▶ **Theorem 1.** *We can compute a largest rectangle contained in a simple polygon with n vertices, possibly with holes, in $O(n^3 \log n)$ time using $O(kn^2)$ space, where k is the number of reflex vertices. We can report all largest rectangles in the same time using $O(n^3)$ space.*

▶ **Theorem 2.** *We can find a largest rectangle in a convex polygon P with n vertices in $O(n^3)$ time using $O(n)$ space.*

Due to lack of space, some proofs and details are omitted.

2 Preliminary

Let P be a simple polygon with n vertices in the plane. For ease of description, we assume that P has no hole. When P has holes, our algorithm works with a few additional data structures and procedures for testing if candidate rectangles contain a hole. We discuss this in Section 8. Without loss of generality, we assume that the vertices of P are given in order

[1] A rectangle is c-fat if its aspect ratio is at most c for some constant c.

along the boundary of P. We denote by k with $k \geq 1$ the number of reflex vertices of P. We assume the general position condition that no three vertices of P are on a line. Whenever we say a *largest rectangle*, it refers to a maximum-area rectangle contained in P.

We use the xy-Cartesian coordinate system and rotate the xy-axes around the origin while the polygon is stationary. We use C_θ to denote the coordinate axes obtained by rotating the xy-axes of the standard xy-Cartesian coordinate system by θ degree counterclockwise around the origin. For a point p in the plane, we use p_x and p_y to denote the x- and y-coordinates of p with respect to the coordinate axes, respectively. We say a segment or line is *horizontal* (or *vertical*) if it is parallel to the x-axis (or the y-axis). Let $\eta(p)$, $\lambda(p)$ and $\delta(p)$ denote the ray (segment) emanating from p going horizontally leftwards, rightwards and vertically downwards in the coordinate axes, respectively, until it escapes P for the first time. We call the endpoint of a ray other than its source point the *foot* of the ray. We denote the foot of $\eta(p)$, $\lambda(p)$ and $\delta(p)$ by $\bar{\eta}(p)$, $\bar{\lambda}(p)$ and $\bar{\delta}(p)$, respectively.

We use $D_\varepsilon(p)$ to denote the disk centered at a point p with radius $\varepsilon > 0$. For any two points p and q in the plane, we use pq and to denote the line segment connecting p and q, and $|pq|$ to denote the length of pq. For a segment s, we use $D(s)$ to denote the smallest disk containing s. For a subset $S \subseteq P$, we define the *visibility region* of S as $V(S) = \{x \in P \mid px \subset P \text{ for every point } p \in S\}$. For a point $p \in P$, we abuse the notation such that $V(p) = V(\{p\})$. For a set X, we use ∂X to denote the boundary of X.

2.1 Existence of a maximum-area rectangle in a simple polygon

The set $\mathcal{G}$ of all parallelograms in the plane is a metric space under the Hausdorff distance measure d_H. The Hausdorff distance between two sets A and B of points in the plane is defined as $d_H(A,B) = \max\{\sup_{a\in A}\inf_{b\in B} d(a,b), \sup_{b\in B}\inf_{a\in A} d(a,b)\}$, where $d(a,b)$ denotes the distance between a and b of the underlying metric. Since the area function $\mu : \mathcal{G} \to \mathbb{R}^{\geq 0}$ is continuous in $\mathcal{G}$, the following lemma assures the existence of a largest rectangle contained in P and thus justifies the problem. Let $\mathcal{R}$ denote the set of all rectangles contained in P. Clearly, $\mathcal{R} \subset \mathcal{G}$.

▶ **Lemma 3.** *The set $\mathcal{R}$ is compact.*

Proof. Define $f : \mathbb{R}^6 \to \mathcal{G}$ to be a function that maps a triplet (p,u,v) of points p, u, and v in $\mathbb{R}^2$ to the parallelogram in $\mathbb{R}^2$ that has p, $p+u$, $p+v$, and $p+u+v$ as the four corners.

If a parallelogram $G \in \mathcal{G}$ is not contained in P, there always exists a point $q \in G$ and a disk $D_\varepsilon(q)$ for some $\varepsilon > 0$ satisfying $D_\varepsilon(q) \cap P = \emptyset$ in the plane. Then for any parallelogram $Q \in \mathcal{G}$ with $d_H(G,Q) < \varepsilon$, the intersection $Q \cap D_\varepsilon(q)$ is not empty and Q is not contained in P. Thus, $\mathcal{C} = \{G \in \mathcal{G} \mid G \not\subset P\}$ is open in $\mathcal{G}$, and therefore $W_P = \big\{(p,u,v) \in \mathbb{R}^6 \mid f(p,u,v) \subset P\big\} = f^{-1}(\mathcal{G} \setminus \mathcal{C})$ is closed. This implies that $T_P = \big\{(p,u,v) \in \mathbb{R}^6 \mid f(p,u,v) \subset P, \text{a rectangle}\big\} = W_P \cap \{(p,u,v) \mid u \cdot v = 0\}$ is closed and also bounded in $\mathbb{R}^6$, i.e. compact. Now we can conclude that $f(T_P) = \mathcal{R}$ is also compact by f being continuous in $\mathbb{R}^6$. ◀

2.2 Classification of largest rectangles

We give a classification of largest rectangles based on the sets of contacts they have on their boundaries with the polygon boundary. We say a rectangle contained in P has a *side-contact* (sc for short) if a side has a reflex vertex of P lying on it, excluding the corners. Similarly, we say a rectangle contained in P has a *corner-contact* (cc for short) if a corner lies on an edge or a vertex of P. When two opposite corners (or two opposite sides) have corner-contacts (or side-contacts), we say the contacts are *opposite*.

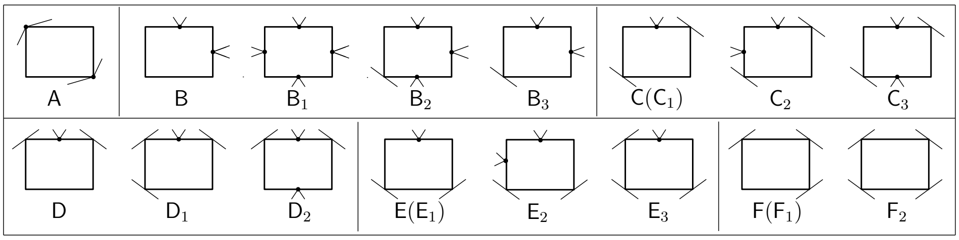


Figure 1 Classification of the determining sets of contacts of largest rectangles when rotations are allowed. Each canonical type X, except A, has a few subtypes X_i for $i = 1, 2, 3$.

Daniels et al. [8] studied this problem with the restriction that rectangles must be axis-aligned. They presented a classification of determining sets of contacts, defined below, into five types for a largest axis-aligned rectangle contained in a simple polygon in the plane.

▶ **Definition 4** (Determining set of contacts [8]). *A set Z of contacts is a determining set of contacts if the largest axis-aligned rectangle satisfying Z has finite area and the largest axis-aligned rectangle satisfying any proper subset of Z has greater or infinite area.*

In our problem, a largest rectangle R is not necessarily axis-aligned. Consider two orthogonal lines which are parallel to the sides of R and pass through the origin. Since R is aligned to the coordinate axes defined by the lines, it also has a determining set of contacts defined by Daniels et al. From this observation, we present a classification of the determining sets of contacts (DS for short) for a largest rectangle in P into six *canonical types*, from A to F, and their subtypes. The classification is given below together with the figures in Figure 1.

- Type A. Exactly two opposite ccs lying on convex vertices of P.
- Type B. One sc on each side incident to a corner c. In addition, B_1 has a sc on each of the other two sides, and B_2 and B_3 have a cc on the corner c' opposite to c. B_2 has another sc on a side incident to c'.
- Type C (C_1). Two ccs on opposite corners c and c', and a sc on a side e incident to a corner c. C_2 has another sc on the side incident to c' and adjacent to e, and C_3 has another sc on a side opposite to e.
- Type D. A sc on a side e and a cc on each endpoint of e. D_1 has another cc and D_2 has another sc on the side opposite to e.
- Type E (E_1). A sc on a side e and a cc on each endpoint of the side e' opposite to e. E_2 has another sc on a side other than e and e'. E_3 has another cc on an endpoint of e.
- Type F (F_1). ccs on three corners. F_2 has ccs on all four corners.

This classification is the same as the one by Daniels et al., except for types A, E, and F. We subdivide the last type in the classification by Daniels et al. into two types, E and F, for ease of description. A DS for type A consists of exactly two opposite corner-contacts lying on convex vertices of P while the corresponding one by Daniels et al. [8] has two opposite corners lying on the boundary (not necessarily on vertices) of P. This is because there is no restriction on the orientation of the rectangle.

2.3 Maximal and breaking configurations

Recall that $\mathcal{R}$ denotes the set of all rectangles contained in P. Our algorithm finds a largest rectangle in $\mathcal{R}$ of each (sub)type so as to find a largest rectangle contained in P. We call a rectangle that gives a *local maximum* of the area function μ among rectangles in $\mathcal{R}$ a *local* *maximum rectangle* (LMR for short). We say an LMR R is of type X if R has all contacts of subtype X_i for some $i = 1, 2, \ldots$. Since a largest rectangle contained in P is a rectangle aligned to the axes that are parallel to its sides, it has contacts of at least one type defined above and is an LMR of that type. Therefore, our algorithm finds a largest rectangle among all possible LMRs of each type.

Consider a rectangle $R \in \mathcal{R}$ that satisfies a DS Z. If there is no contact other than Z, there exists a continuous transformation of R such that the transformed rectangle is a rectangle contained in P and satisfying Z. Then by such a continuous transformation the area of R may change. Imagine we continue with such a transformation until the transformed rectangle R' gets another contact. In this case, we say R' is in a *breaking configuration* (BC for short) of Z. During the transformation, the area of R' may become locally maximum. If R' is locally maximum and has no contact other than Z, we say R' is in a *maximal configuration* of Z. There can be $O(1)$ maximal configurations of Z, which can be observed from the area function of the rectangle.

▶ **Lemma 5.** *An LMR satisfying Z is in a maximal configuration or a breaking configuration.*

For a breaking configuration Z' of a DS Z, observe that $Z' \setminus Z$ is a singleton and Z' can be a BC of some other DSs. With this fact, we can classify BCs by avoiding repetition and reducing them up to symmetry. See Figure 4 for breaking configurations.

We use $\Gamma_\theta(Z)$ to denote the axis-aligned rectangle of largest area that satisfies a DS Z in C_θ. We say a DS Z is *feasible* at an orientation θ if $\Gamma_\theta(Z)$ is a rectangle contained in P. We say an orientation θ is *feasible* for Z if $\Gamma_\theta(Z)$ is contained in P.

3 Computing a largest rectangle of type A

It suffices to check all possible squares in P with two opposite corners on convex vertices of P. Since ∂P is a simple closed curve, we can determine if a rectangle R is contained in P by checking if all four sides of R are contained in P.

▶ **Lemma 6.** *We can compute a largest rectangle among all LMRs of type A in $O((n-k)n + (n-k)^2 \log n)$ time using $O(n)$ space.*

4 Computing a largest rectangle of type B

We show how to compute all LMRs of type B and a largest rectangle among them. We compute for each DS a largest LMR over the maximal and breaking configurations. In doing so, we maintain a combinatorial structure for each reflex vertex which helps compute all LMRs of type B during the rotation of the coordinate axes.

4.1 Staircase of a point in a simple polygon

We define the staircase $\mathsf{S}(u)$ of a point $u \in P$ as the set of points $p \in P$ with $p_x \leq u_x$ and $p_y \leq u_y$ such that the axis-aligned rectangle with diagonal up is contained in P but no axis-aligned rectangle with diagonal uq is contained in P for any point $q \in P$ with $q_x < p_x$ and $q_y < p_y$. Thus, $\mathsf{S}(u)$ can be represented as a chain of segments. See Figure 2 (a).

The staircase of a point $u \in P$ in orientation θ, denoted by $\mathsf{S}_\theta(u)$, is defined as the staircase of u in C_θ. Every axis-aligned segment of $\mathsf{S}_\theta(u)$ has one endpoint at a vertex of P, $\bar{\eta}(u)$, or $\bar{\delta}(u)$. A segment of $\mathsf{S}_\theta(u)$ that is not aligned to the axes is a part of an edge e of P and is called an *oblique segment*. (We say e appears to the staircase in this case.)

Each vertex of $\mathsf{S}_\theta(u)$ which is a vertex P, $\bar{\eta}(u)$, or $\bar{\delta}(u)$ is called an *extremal vertex.* An extremal vertex v is called a *tip* if it is a reflex vertex of P. A vertex of $\mathsf{S}_\theta(u)$ contained in P is called a *hinge.* A *step* of $\mathsf{S}_\theta(u)$, denoted by an ordered pair (a, b), is the part of $\mathsf{S}_\theta(u)$ between two consecutive extremal vertices a and b along $\mathsf{S}_\theta(u)$, where $a_x \leq b_x$ and $a_y \geq b_y$. It consists of either (i) two consecutive segments ar and rb for $r = \delta(a) \cap \eta(b)$, or (ii) three consecutive segments $a\bar{\delta}(a)$, $\bar{\delta}(a)\bar{\eta}(b)$, and $\bar{\eta}(b)b$ with $\bar{\delta}(a)_x \leq \bar{\eta}(b)_x$ and $\bar{\delta}(a)_y \geq \bar{\eta}(b)_y$. Note that $\bar{\delta}(a)\bar{\eta}(b)$ is the oblique segment of step (a, b) which we denote by $\mathsf{ob}(a, b)$. A horizontal, vertical, or oblique segment of a step can be just a point in case of degeneracy.

We can construct $\mathsf{S}_\theta(u)$ for a fixed θ in $O(n)$ time by traversing the boundary of P in counterclockwise direction starting from $\bar{\eta}(u)$ while maintaining the staircase of u with respect to the boundary chain traversed so far. When the next vertex v of the boundary chain satisfies $v_x \geq t_x$ and $v_y \leq t_y$ for the last vertex t of the current staircase, we append it to the staircase. If (part of) the edge incident to v is an oblique segment of the staircase, then we append it together with v to the staircase. If $v_x < t_x$, we ignore v and proceed to the vertex next to v. If $v_x \geq t_x$ and $v_y > t_y$, we remove the portion of the current staircase violated by v and append v to the staircase accordingly. Observe that each vertex and each edge appear on the staircase at most once during the construction.

4.2 Maintaining the staircase during rotation of the coordinate system

Bae et al. [4] considered the rectilinear convex hull for a set Q of n point in the plane and presented a method of maintaining it while rotating the coordinate system in $O(n^2)$ time. The boundary of the rectilinear convex hull consists of four maximal chains, each of which is monotone to the coordinate axes. We adopt their method and maintain the staircase of a reflex vertex u in a simple polygon.

The combinatorial structure of $\mathsf{S}_\theta(u)$ changes during the rotation. Figure 2 (b-f) show $\mathsf{S}_0(u)$, $\mathsf{S}_{\theta_1}(u)$ and $\mathsf{S}_{\theta_2}(u)$ for three orientations $0, \theta_1$, and θ_2 ($0 < \theta_1 < \theta_2 < \pi/2$). Two consecutive steps, (a, b) and (b, c), of the staircase merge into one step (a, c) when $\bar{\delta}(a)$ meets b (Figure 2 (b)). A step (a, b) splits up into two steps (a, v) and (v, b) when $\bar{\eta}(b)$ meets a polygon vertex v (Figure 2 (c)). A step changes its type between (A) and (B) when the hinge of a step hits a polygon edge (and then it is replaced by an oblique segment) or the oblique segment of a step degenerates to a point (and then it becomes a hinge) (Figure 2 (d)). The upper tip a (or the lower tip b) of a step (a, b) can disappear from the staircase when $\bar{\delta}(\bar{\eta}(u))$ meets a (or $\bar{\eta}(\bar{\delta}(u))$ meets b). Finally, a vertex, possibly along with an edge incident to it, can be added to or deleted from $\mathsf{S}_\theta(u)$ when it is met by $\bar{\eta}(\bar{\delta}(u))$ or $\bar{\delta}(\bar{\eta}(u))$. We call such a change of the staircase due to the cases described above a *step event.*

One difference of the staircase $\mathsf{S}_\theta(u)$ to the one for a point-set by Bae et al. is that the two boundary points of $\mathsf{S}_\theta(u)$ are $\bar{\eta}(u)$ and $\bar{\delta}(u)$. Since the polygon is not necessarily monotone with respect to the axes, the staircase may change discontinuously when $\bar{\eta}(u)$ or $\bar{\delta}(u)$ meets a vertex of P, which we call a *ray event.* The step of $\mathsf{S}_\theta(u)$ incident to $\bar{\eta}(u)$ is replaced by a chain of $O(n)$ steps when $\bar{\eta}(u)$ meets a vertex of P (Figure 2 (e)). A subchain incident to $\bar{\delta}(u)$ is replaced by a single step when $\bar{\delta}(u)$ meets a vertex of P (Figure 2 (f)). We call the appearance or disappearance of a step caused by a ray event a *shift event* of the ray event. Note that $O(n)$ shift events occur at a ray event. Observe that all the changes occurring in a staircase during the rotation are caused by step, ray, or shift events. We abuse $\mathsf{S}_\theta(u)$ to denote the combinatorial structure of the staircase if understood in context.

▶ **Lemma 7.** *The number of events that occur to $S_\theta(u)$ during the rotation is $O(n^2)$.*

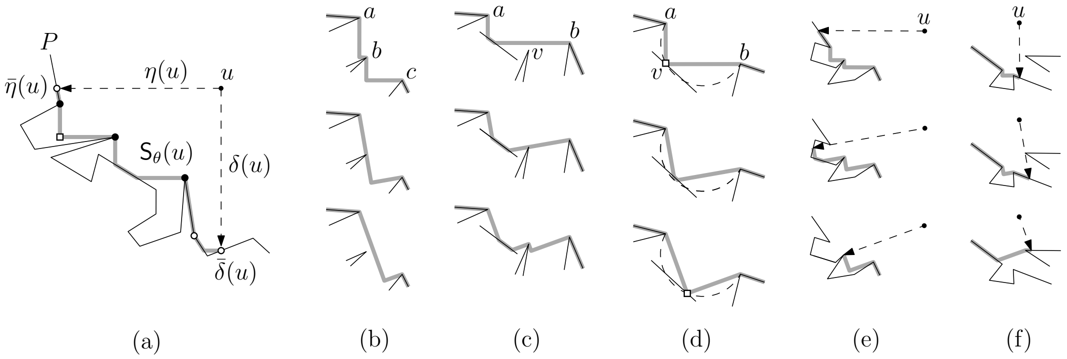


Figure 2 (a) Staircase $S_\theta(u)$ (thick gray chain) and the tips (black disks), the extremal vertices (black disks and circles), and the hinge (square) of $S_\theta(u)$. (b–d) Step events, and (e–f) ray events during the rotation of the coordinate system.

To capture these combinatorial changes and maintain the staircase during the rotation, we construct for every reflex vertex p of P, the list of segments of visibility region $V(p)$ sorted in angular order. We compute for every pair (p, q) of reflex vertices of P, the list $C(p, q)$ of vertices and segments of $\partial V(\{p, q\}) \cap D(pq)$, sorted in angular order with respect to p and q. We also compute for every pair (p, e) of a reflex vertex p and edge e, the sorted list $L(p, e)$ of angles at which $\delta(\bar{\eta}(p))$ or $\eta(\bar{\delta}(p))$ meets a vertex of P while $\bar{\eta}(p)$ or $\bar{\delta}(p)$ lies on e. We store for each orientation in $L(p, e)$ the information on the vertex corresponding to the orientation. This can be computed by finding the points that e intersects with the boundary of $D(tp)$ for each vertex t of P. These structures together constitute the *event map*.

We also construct an *event queue* for each reflex vertex, which is a priority queue that stores events indexed by their orientations. This is to update the staircase during the rotation in a way similar to the one by Bae et al. [4] using the event map. The event map is of size $O(kn^2)$ and can be constructed in $O(kn^2 \log n)$ time. For a reflex vertex u, we maintain $\mathsf{S}_\theta(u)$ and the event queue $\mathcal{Q}$ for u during the rotation using the event map. We also store the extremal vertices and edges of the staircase in a balanced binary search tree $\mathcal{T}$ representing $\mathsf{S}_\theta(u)$ in order along the staircase so as to insert and delete an element in $O(\log n)$ time.

▶ **Lemma 8.** *The staircases of all k reflex vertices of P can be constructed and maintained in $O(kn^2 \log n)$ time using $O(kn^2)$ space during the rotation.*

4.3 Data structures – double staircases, event map, and event queue

Our algorithm computes all LMRs of type B during the rotation and returns an LMR with largest area among them. To do so, it maintains for each reflex vertex u two staircases, $S_\theta(u)$ and $S_{\theta+\frac{\pi}{2}}(u)$ which we call the *double staircase* of u, during the rotation of the coordinate axes and computes the LMRs of type B that have u as the top sc. Let I denote the interval of orientations such that the horizontal line with respect to any $\theta \in I$ passing through u is tangent to the boundary of P locally at u. Let R_θ be the largest axis-aligned rectangle of type B in $\theta \in I$ that is contained in P and has u as the top sc. Observe that every reflex vertex lying on the right side is a tip of $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$. We use X to denote the contact set around the bottom-left corner c_θ of R_θ. Then X contains (1) a tip of $\mathsf{S}_\theta(u)$ touching the left side and a tip on either $\mathsf{S}_\theta(u)$ or $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ touching the bottom side (type B_1), (2) an oblique segment e on $\mathsf{S}_\theta(u)$ touching c_θ and a tip on either $\mathsf{S}_\theta(u)$ or $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ touching the bottom side (type B_2), or (3) just an oblique segment e on $\mathsf{S}_\theta(u)$ touching c_θ (type B_3).

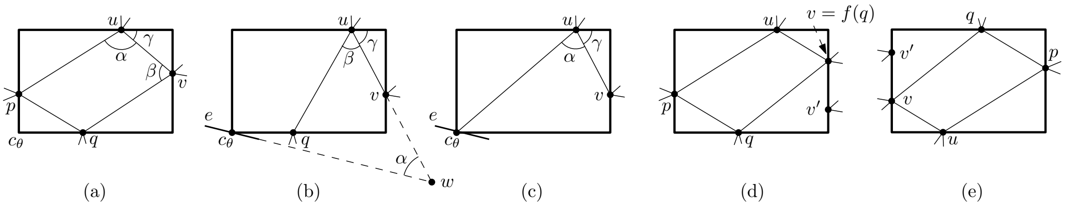


Figure 3 LMRs of (a) type B_1, (b) type B_2, and (c) type B_3. (d) Step event on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ that $f(q)$ changes from v to v'. (e) The step event in (d) is a step event on $\mathsf{S}_{\theta+\pi}(q)$.

For a reflex vertex u of P, we construct the double staircase of u, $\mathsf{S}_0(u)$ and $\mathsf{S}_{\frac{\pi}{2}}(u)$. Then we maintain the event queue $\mathcal{Q}$ containing event orientations in order: the orientations for staircase events (step and ray events) defined in previous section and the orientations at which two vertices of $V(u)$ are aligned horizontally. The set of the orientations of the latter type is to capture the event orientations at which a tip of $\mathsf{S}_\theta(u)$ is aligned horizontally with a tip of $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$. We call them *double staircase events.* We initialize $\mathcal{Q}$ with the latter type events. Note that it does not increase the time and space complexities of the event queue.

4.4 Computing LMRs of type B_1

Consider a reflex vertex t of P that appears as a tip on $\mathsf{S}_\theta(u)$. We use $f(t)$ to denote the upper tip of the step on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ aligned horizontally to t. For example, in Figure 3(a,b), $v = f(q)$ in $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$. During the rotation of the coordinate axes, we consider the change of $f(t)$ for each tip t on $\mathsf{S}_\theta(u)$, as well as step, ray, and shift events on the double staircase. At each orientation, $f(t)$ can be computed in $O(\log n)$ time via binary search on $S_{\theta+\frac{\pi}{2}}(u)$ with t_y since the staircase chain is monotone with respect to the y-axis. Thus we do not need to save the value $f(t)$ for each tip t. We consider the orientation when a tip t on $\mathsf{S}_\theta(u)$ and $f(t)$ on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ are aligned horizontally so that $f(t)$ is set to the next tip on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$. At such a orientation, we detect a DS candidate of the type B_1 with top, left, bottom, and right SCs as u, p, q, and $v = f(q)$, respectively. Note that the bottom SC q might have $q_x > u_x$. We process only the case that q is a tip on $\mathsf{S}_\theta(u)$, since the other case can be handled when fixing p as an upper side contact, as described in the following, when step (q, v) disappears by a step event on $\mathsf{S}_{\theta+\frac{\pi}{2}}(p)$ and u is a tip on $\mathsf{S}_{\theta+\pi}(p)$.

Consider an event E occurring at θ. When a step of the double staircase that possibly contributes to a DS of type B_1 changes due to E, we detect possible DSs that have been associated with it. Consider a DS $\{u, p, q, v\}$ as in Figure 3(a). If E is a step or shift event on $\mathsf{S}_\theta(u)$, $O(1)$ tips appear or disappear on $\mathsf{S}_\theta(u)$ and $O(1)$ DSs are detected at each such event. If E is a step or shift event on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$, there are $O(n)$ tips t on $\mathsf{S}_\theta(u)$ such that $f(t)$ changes. Observe that such an event corresponds to a step event on the staircase of q in $\mathsf{C}_{\theta+\pi}$. See Figure 3(d-e). Thus, we may consider only step and shift events on $\mathsf{S}_\theta(u)$ together with the double staircase events to detect possible DSs of type B_1.

We consider $O(1)$ DSs for each event. Let $Z = \{u, p, q, v\}$ be a DS of type B_1 such that (p, q) is changed or q and v are aligned horizontally by an event E at θ_E. Given a closed interval J, we can compute the set Θ_Z of orientations $\theta_Z \in J$ maximizing $\mu(\Gamma_\theta(Z))$ locally in $O(1)$ time because the area function $\mu(\Gamma_\theta(Z)) = (|uv|\cos\gamma + |up|\cos(\pi - (\alpha+\gamma)))(|uv|\sin\gamma + |qv|\cos(\frac{\pi}{2} - (\beta - \gamma)))$ has $O(1)$ extremal values in J. For angles α, β, γ, see Figure 3(a).

We find the maximal interval J of orientations for $Z = \{u, p, q, v\}$ found in an event E occurring at θ_E such that $\theta_E \in J$ and all elements of Z appear on the double staircase of u. This can be done by maintaining the latest orientation ($< \theta_E$) at which (p, q) starts to appear as a step, the latest orientation ($< \theta_E$) at which v starts to appear as a tip to the double staircase, and the orientation at which $f(q)$ was set to v. Then $J = [\theta_a, \theta_E]$, where θ_a is the latest of orientation at which all elements of Z and the step consisting of elements of Z start to appear on the double staircase while satisfying $f(q) = v$. Note that the LMRs with contact Z occur at every orientation of Θ_Z and the two endpoints (orientations) of J. Observe that the rectangle $\Gamma_{\theta_Z}(Z)$ with $\theta_Z \in \Theta_Z$ corresponds to a maximal configuration, and $\Gamma_\theta(Z)$ with θ being an endpoint of J corresponds to a breaking configuration. In this way, we can compute $O(1)$ LMRs satisfying Z in $O(1)$ time.

For a reflex vertex u, we maintain an event queue $\mathcal{Q}$. For each event E in $\mathcal{Q}$, our algorithm finds $O(1)$ DSs Z that become infeasible by E, and computes the LMRs in $O(1)$ time. Observe that a DS Z of type B_1 becomes infeasible only at shift, step, and double staircase events. Since our algorithm is applied to every reflex vertex u of P, we do not need to process the shift and step events occurring on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$. Therefore, we can detect every possible DS Z by processing the events in Q. By Lemma 8, we have the following lemma.

▶ **Lemma 9.** *Our algorithm computes all LMRs of type B_1 with largest area in $O(kn^2 \log n)$ time, where k is the number of reflex vertices.*

4.5 Computing LMRs of type B_2

A DS $Z = \{u, e, q, v\}$ of type B_2 consists of three reflex vertices u, q, v realizing the top, bottom, right sc and an oblique segment e realizing the cc at the bottom-left corner c_θ of $\Gamma_\theta(Z)$. Let w be the point where the extended line of e and the line through u and v cross. If w appears below c_θ, then the area function is $\mu(\Gamma_\theta(Z)) = |uq|\sin(\beta+\gamma)\big(\cot(\gamma-\alpha)(|uw|\sin\gamma - |uq|\sin(\beta+\gamma)) - |vw|\cos\gamma\big)$. See Figure 3(b). The area of $\Gamma_\theta(Z)$ with w appearing above u can be computed in a similar way. Note that there are $O(1)$ orientations that maximize $\mu(\Gamma_\theta(Z))$ locally and they can be computed in $O(1)$ time.

Observe that q is contained in $\mathsf{S}_\theta(u)$ or (q, v) is a step on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$. In addition to the method from the Section 4.4, we also handle the case that (q, v) is a step on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ as follows. For a reflex vertex t appearing as a tip on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$, let $g(t)$ be the edge that contains $\bar{\eta}(t)$. We consider every change of $f(t)$ for each tip t on $\mathsf{S}_\theta(u)$ and the every change of $g(t)$ for each tip t on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ during the rotation.

Consider an event E occurring at θ_E. If $f(q)$ changes from v to v', we detect the DS $\{u, e, q, v\}$, where e is the edge containing the oblique segment of the step q belongs to, in a similar way as we process such an event of type B_1.

So it remains to consider the case for an event E that changes $g(q)$ for each tip q on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ at θ_E. Observe that $g(q)$ changes only if a double staircase event occurs associated with q. Whenever a new step (q, v) appears on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$, we do binary search for $\bar{\eta}(q) \in e$ in $V(q)$. When $g(q)$ changes or a step (q, v) disappears caused by a step or a shift event on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$, we find $O(1)$ LMRs with DS $Z = \{u, e, q, v\}$, and check if they are in P by checking if the boundary of the rectangles are in P, since we do not know if e appears on $\mathsf{S}_\theta(u)$ or not. These LMRs can be computed in a similar way as we do for type B_1. Together with Lemma 8, we have the following lemma.

▶ **Lemma 10.** *Our algorithm computes all LMRs of B_2 with the largest area in $O(kn^2 \log n)$ time, where k is the number of reflex vertices of P.*

4.6 Computing LMRs of type B_3

Consider the case when DS $Z = \{u, e, v\}$ is feasible, where e is a polygon edge that appears as an oblique segment on $\mathsf{S}_\theta(u)$ and v is a tip on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ (type B_3). See Figure 3(c). To achieve the largest area, we observe that the bottom-left corner of LMRs satisfying Z must lie at the midpoint c of the extended line segment pq of e, where p and q are the intersection points of the line containing e with $\eta(u)$ and $\delta(v)$, respectively. The area function of $\Gamma_\theta(\{u, c_\theta, v\})$, the rectangle with top sc on u, bottom-left cc on c_θ, and right sc on v, is convex with respect to $c_\theta \in l$, where l is the line containing e. If c does not lie on e, c_θ must lie on a point of e closest to c to maximize the rectangle area.

The area function of $\Gamma_\theta(Z)$ for a DS Z of B_3 is $\mu(\Gamma_\theta(Z)) = |uc_\theta| \sin(\alpha+\gamma)\big(|uc_\theta| \cos(\pi - (\alpha+\gamma)) + |uv| \cos\gamma\big)$, where c_θ is the midpoint of pq (if the midpoint lies on e) or the endpoint of e that is closer to the midpoint (otherwise) at θ. Since the midpoint moves along l in one direction as θ increases, there are $O(1)$ intervals of orientations at which the midpoint of pq is contained in e, and thus this area function has $O(1)$ extremal values in I.

Our algorithm for computing all LMRs of type B_3 is simple. First we fix the top sc on u. For each pair of an edge e and a reflex vertex v, we compute the set Θ_Z of orientations that maximize $\mu(\Gamma_\theta(Z))$ locally for $Z = \{u, e, v\}$. Observe that Θ_Z consists of $O(1)$ orientations because the area function has $O(1)$ extremal values. Then for each $\theta_Z \in \Theta_Z$, we find two orientations θ_1, θ_2 closest to θ_Z with $\theta_1 \leq \theta_Z$ and $\theta_2 \geq \theta_Z$ such that the top-right corner of $\Gamma_\theta(Z)$ is contained in P by applying binary searching on $C(u, v)$. Finally, we check if $\Gamma_\theta(Z)$ is contained in P for $O(1)$ such orientations θ by checking if the boundary of the rectangles are contained in P. This way we can compute all LMRs of B_3 with top sc on u. See Figure 4. By using the event map and Lemma 8, we have the following lemma.

▶ **Lemma 11.** *Our algorithm computes all LMRs of B_3 with largest area in $O(kn^2 \log n)$ time, where k is the number of reflex vertices of P.*

5 Computing a largest rectangle of types C and D

LMRs of types C and D can be computed in a way similar to the one for type B. For each reflex vertex u, we find all LMRs of types C and D that have u on its top side while maintaining the double staircase of u.

▶ **Lemma 12.** *We can compute a largest rectangle among all LMRs of types C and D in $O(kn^2 \log n)$ time using $O(kn^2)$ space, where k is the number of reflex vertices of P.*

6 Computing a largest rectangle of type E

We consider the LMRs of type E. Let u be a reflex vertex of P. We detect every DS Z of type E, containing $\{u, e_l, e_r\}$, where u is the top sc, e_l the bottom-left cc, and e_r the bottom-right cc. Observe that for each LMR satisfying Z, e_l and e_r appear as oblique segments $\mathsf{ob}(p, q) \subset e_l$ and $\mathsf{ob}(t, v) \subset e_r$ of $\mathsf{S}_\theta(u)$ and $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$, respectively, such that $\bar{\eta}(t) \in \mathsf{ob}(p, q)$ or $\bar{\lambda}(q) \in \mathsf{ob}(t, v)$, depending on whether $q_y \leq t_y$ or not. Using this fact, we detect the events at which Z becomes infeasible, and compute the LMRs satisfying Z.

We compute LMRs of type E at (1) every step and shift event (and ray event) with a step containing an oblique segment on the double staircase, and (2) every event such that $\bar{\lambda}(q)$ meets $\bar{\delta}(t)$ on an edge e for a tip q of $\mathsf{S}_\theta(u)$ and a tip t of $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$. At an event E of case (1), we find DSs Z that become infeasible caused by E. At an event E of case (2) occurring at θ_E, we have a step (p, q) on $\mathsf{S}_\theta(u)$ and a step (t, v) on $\mathsf{S}_{\theta+\frac{\pi}{2}}(u)$ such that $\bar{\lambda}(q)$ meets $\bar{\delta}(v)$

on an edge of P. We consider the same DSs Z_1 and Z_2 considered in case (1). Observe that E corresponds to the step event of the double staircase of v at $\theta_E - \frac{\pi}{2}$. The double staircase of v has a step event at $\theta_E - \frac{\pi}{2}$ that $\bar{\delta}(\bar{\lambda}(v))$ meets q (equivalently, $\bar{\lambda}(q)$ meets $\bar{\delta}(v)$ on an edge of P at θ_E). Thus, we can capture E by maintaining the double staircase of v and insert E to the event queue of u in $O(\log n)$ time. We compute this type of events for all reflex vertices of P by maintaining double staircases of the reflex vertices of P and insert the events to the event queues of their corresponding reflex vertices whenever such events are found. There are $O(kn^2)$ events in total, and they can be found and inserted to the event queues in $O(kn^2 \log n)$ time.

Whenever detecting a DS Z, we take a closed interval J of orientations at which Z is possibly feasible, and compute $O(1)$ LMRs with contact Z within J. When Z contains $\{u, e_l, e_r\}$ as the top sc u, bottom-left cc e_l, and bottom-right cc e_r, J is the interval such that $\mathsf{ob}(p,q) \subseteq e_l$, $\mathsf{ob}(t,v) \subseteq e_r$, and $\bar{\lambda}(\bar{\delta}(p)) \in \mathsf{ob}(t,v)$ or $\bar{\lambda}(q) \in \mathsf{ob}(t,v)$. Note that the interval satisfying $\bar{\lambda}(\bar{\delta}(p)) \in \mathsf{ob}(t,v)$ or $\bar{\lambda}(q) \in \mathsf{ob}(t,v)$ can be computed in $O(\log n)$ time using binary search on $L(p, e_l)$ and $L(v, e_r)$. If Z contains p as the left sc or e' as the top-left cc, we consider the BCs such that v is the right sc or there is another cc on the top-right corner. Note that the BC of the second case corresponds to a BC of type D_1. The orientation at which such a BC occurs can be computed in $O(1)$ time by solving basic system of linear equations. Therefore, J can be computed in $O(\log n)$ time.

We compute $O(1)$ LMRs for each event and check if they are contained in P in $O(\log n)$ time. There are $O(kn^2)$ events corresponding to case (2) which are computed in $O(kn^2 \log n)$ time before we handle the events of type E. By Lemma 7, we have the following lemma.

▶ **Lemma 13.** *We can compute a largest rectangle among all LMRs of type E in $O(kn^2 \log n)$ time using $O(kn^2)$ space, where k is the number of reflex vertices of P.*

7 Computing a largest rectangle of type F

To find all LMRs of type F, we compute the maximal configurations and breaking configurations of DSs of type F as follows. Consider a DS $Z_1 = \{e_1, e_l, e_2\}$ of type F_1 and a DS $Z_2 = \{e_1, e_l, e_r, e_2\}$ of type F_2. Then the LMR of a BC of type F_1 is the rectangle satisfying $Z_1 \cup \{u\}$ or $Z_1 \cup \{e_r\}$, or a rectangle satisfying Z_1 with cc on an end vertex of an edge in Z_1, where u is a reflex vertex and e_r is an edge of P. The LMR satisfying $Z_1 \cup \{u\}$ belongs to type D_1 or E_3 which is computed as an LMR of D or E. The LMR satisfying $Z_1 \cup \{e_r\}$ belongs to type F_2 and it is considered for type F_2. The LMR of a BC of type F_2 is the rectangle satisfying $Z_2 \cup \{u\}$, $Z_2 \cup \{e'\}$ or the rectangle satisfying Z_2 with cc on an end vertex of an edge in Z_2, where u is a reflex vertex and e' is an edge of P. The LMR satisfying $Z_2 \cup \{u\}$ belongs to a BC of type E_3 which is computed as an LMR of type E. (See the last BC of type E_3 in Figure 4.) Thus, we consider the LMRs of maximal configurations of type F or breaking configurations Z of type F containing a cc on an end vertex of an edge in Z only.

We say an edge pair (e_1, e_2) is *h-aligned* (and *v-aligned*) at θ if there are points $p_1 \in e_1$ and $p_2 \in e_2$ such that p_1p_2 is horizontal (and vertical) and is contained in P at θ. A pair (e_1, e_2) of edges is *h-misaligned* (and *v-misaligned*) at θ if the pair is not h-aligned (and not v-aligned) at θ. Note that a edge pair (e_1, e_2) changes between being h- or v-aligned and being h- or v-misaligned only when two vertices of P are aligned horizontally or vertically during the rotation. We say a triplet (e_1, e_l, e_2) of edges *t-aligned* at θ if there is a point $x \in e_1$ such that $\bar{\lambda}(x) \in e_2$ and $\bar{\delta}(x) \in e_l$ at some $\theta' \in \{\theta, \theta + \frac{\pi}{2}, \theta + \pi, \theta + \frac{3\pi}{2}\}$. An edge triplet (e_1, e_l, e_2) is *t-misaligned* if it is not t-aligned at θ.

We compute LMRs of type F at (1) every event such that two vertices are aligned horizontally or vertically, and (2) every event such that $\bar{\delta}(\bar{\eta}(u))$ meets p for every vertex pair (u, p) at $\theta' \in \{\theta, \theta + \frac{\pi}{2}, \theta + \pi, \theta + \frac{3\pi}{2}\}$. In case (1), at an event such that two vertices u and v are aligned horizontally, we find an edge pair (e_1, e_2) which becomes h-misaligned in $O(\log n)$ time using ray-shooting queries with $\eta(u)$ and $\lambda(v)$, assuming that $u_x < v_x$ if such pair exists. Then we also find edges e_l and e_r such that e_l contains $\bar{\delta}(\bar{\eta}(u))$ and e_r contains $\bar{\delta}(\bar{\lambda}(v))$. We can find such edges in $O(\log n)$ time using ray-shooting queries. Then we compute the set Θ_{Z_i} of orientations that maximize $\mu(\Gamma_\theta(Z_i))$ for each DS $Z_1 = \{e_1, e_l, e_2\}$, $Z_2 = \{e_1, e_r, e_2\}$ and $Z_3 = \{e_1, e_l, e_r, e_2\}$ and check if $\Gamma_\theta(Z_i)$ is contained in P for $\theta \in \Theta_{Z_i}$. Observe that the top-left cc of every LMR of type F_1 lies at the midpoint c of wt, for the intersection w of two lines, one containing e_1 and one containing e_l, and the intersection t of two lines, one containing e_1 and one containing e_2. Note that every area function of type F as $O(1)$ extremal values. If $c \notin e_1$, we take the point on e_1 that is closest to the midpoint. Thus, each Θ_{Z_i} has $O(1)$ elements and we can check for each $\Gamma_\theta(Z_i)$ if it is contained in P in $O(\log n)$ using ray-shooting queries. We also compute the BCs satisfying Z_i with cc on an end vertex of an edge in Z_i, and check their feasibility. There are $O(1)$ such BCs which can be computed in $O(1)$ time. We can compute in $O(1)$ time $\mu(\Gamma_\theta(Z))$ for each BC Z. An event at which two vertices u and v are aligned vertically can be handled in a symmetric way.

In case (2), when $\bar{\delta}(\bar{\eta}(u))$ meets p, we find an edge triplet (e_1, e_l, e_2) which becomes t-misaligned in $O(\log n)$ time using ray-shooting queries with $\eta(u), \lambda(u)$ and $\delta(p)$. We also find edges e_r and e'_r such that $\bar{\delta}(\bar{\lambda}(u)) \in e_r$ and $\bar{\lambda}(p) \in e'_r$ in $O(\log n)$ time using ray-shooting queries. Similar to case (1), we compute Θ_{Z_i} for each DS $Z_1 = \{e_1, e_l, e_2\}$, $Z_2 = \{e_1, e_l, e_r, e_2\}$ and $Z_3 = \{e_1, e_l, e'_r, e_2\}$ and check if $\Gamma_\theta(Z_i) \subseteq P$ for $\theta \in \Theta_{Z_i}$. Then we compute the BCs satisfying Z_i with cc on an end vertex of an edge in Z_i and check their feasibility.

There are $O(n^2)$ events corresponding to case (1) and $O(n^3)$ events corresponding to case (2). We can compute them in $O(n^3)$ time in total. For each event, we find $O(1)$ DSs in $O(\log n)$ time and compute $O(1)$ maximal and breaking configurations of each DS in $O(1)$ time, and check their feasibility in $O(\log n)$ time. And the only data structure we use for type F is a ray-shooting data structure of $O(n)$ space.

▶ **Lemma 14.** *We can compute a largest rectangle among all LMRs of type F in $O(n^3 \log n)$ time using $O(n)$ space.*

8 Computing a largest rectangle in a simple polygon with holes

Our algorithm can compute a largest rectangle in a simple polygon P with h holes and n vertices. We use the same classification of largest rectangles and find the LMRs of the six types. We construct a ray-shooting data structure, such as the one by Chen and Wang [7] in $O(n + h^2 \operatorname{polylog} h)$ time using $O(n + h^2)$ space, which supports a ray-shooting query in $O(\log n)$ time. We also construct the visibility region from each vertex of P, which can be done in $O(n^2 \log n)$ time using $O(n^2)$ space by using the algorithm in [7]. Each visibility region is simple and has $O(n)$ complexity. The staircase of a vertex of P can be constructed in $O(n \log n)$ time using plane sweep with ray-shooting queries. Each staircase of a vertex u of P has $O(n)$ space. There are $O(n^2)$ events to the staircase of u since it is equivalent to the staircase constructed in $V(u)$, a simple polygon with $O(n)$ vertices.

We say a rectangle is *empty* if there is no hole contained in it. Since P has holes, there can be a hole contained in a rectangle R even though every side of R is contained in P. Thus, we check the emptiness of rectangles, together with the test for their sides being contained in P. The emptiness of a rectangle can be checked by constructing a triangular range searching data structure for n vertices of P in $O(n^2)$ time and space [10]. For a query with two triangles obtained from subdividing the rectangle by a diagonal, it answers the number of vertices lying in the triangle in $O(\log n)$ time. Since the remaining part of our algorithm works as it is, we have Theorem 1.

9 Computing a largest rectangle in a convex polygon

When P is convex, there is no reflex vertex and therefore it suffices to consider only the LMRs of types A and F. Using the method in Lemma 6, we can compute a largest LMR of type A in $O(n^2 \log n)$ time using $O(n)$ space.

For type F, we find the events considered in Section 7 and all DSs corresponding to the events in case (1) that a vertex u is aligned to another vertex in $O(n)$ time by maintaining rays $\lambda(u)$, $\delta(\bar{\lambda}(u))$, $\delta(u)$ and $\lambda(\bar{\delta}(u))$ during the rotation. Since P is convex, the foot of each ray emanating from u changes *continuously* along the boundary of P. Similarly, we find all DSs corresponding to the events in case (2) in Section 7 that an edge triplet becomes t-misaligned. It is caused by $\bar{\delta}(\bar{\eta}(u))$ meeting p for a vertex pair (u, p) and its corresponding DSs can be computed in $O(n)$ time by maintaining $\eta(u)$, $\delta(u)$, $\xi(p)$ and $\lambda(p)$ during the rotation, where $\xi(p)$ is the vertically upward ray from p. Thus, we can find all events and their corresponding DSs in $O(n^2)$ time for case (1) and in $O(n^3)$ time for case (2). Since every LMR is contained in P, we can find the maximal configuration of each DS in $O(1)$ time. We conclude with Theorem 2.

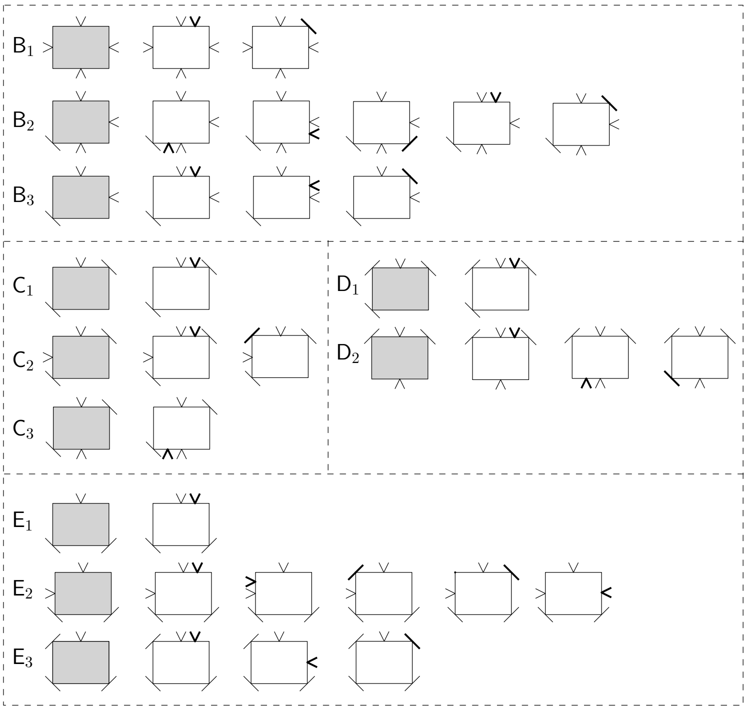


Figure 4 Canonical (sub)types (gray rectangles) and their breaking configurations without duplication. The breaking configurations of subtypes F_1 and F_2 appear as breaking configurations of other types: By adding a sc to a DS Z of type F_1, Z becomes a BC of type either D_1 or E_3. By adding a cc to a DS Z of type F_1, Z becomes a BC of type F_2. By adding a sc to a DS Z of type F_2, Z becomes a BC (of the last type) of type D_1.

References

1 Alok Aggarwal and Joel Martin Wein. Computational Geometry Lecture Notes for MIT, 1988.

2 Helmut Alt, David Hsu, and Jack Snoeyink. Computing the Largest Inscribed Isothetic Rectangle. In *Proceedings of 7th Canadian Conference on Computational Geometry (CCCG 1995)*, pages 67–72. University of British Columbia, 1995.

3 Nina Amenta. Bounded Boxes, Hausdorff Distance, and a New Proof of an Interesting Helly-type Theorem. In *Proceedings of 10th Annual Symposium on Computational Geometry (SoCG 1994)*, pages 340–347, 1994.

4 Sang Won Bae, Chunseok Lee, Hee-Kap Ahn, Sunghee Choi, and Kyung-Yong Chwa. Computing minimum-area rectilinear convex hull and L-shape. *Computational Geometry*, 42(9):903–912, 2009.

5 Ralph P. Boland and Jorge Urrutia. Finding the Largest Axis-Aligned Rectangle in a Polygon in $O(n \log n)$ time. In *Proceedings of 13th Canadian Conference on Computational Geometry (CCCG 2001)*, pages 41–44, 2001.

6 Sergio Cabello, Otfried Cheong, Christian Knauer, and Lena Schlipf. Finding largest rectangles in convex polygons. *Computational Geometry*, 51:67–74, 2016.

7 Danny Z. Chen and Haitao Wang. Visibility and ray shooting queries in polygonal domains. *Computational Geometry*, 48(2):31–41, 2015.

8 Karen Daniels, Victor Milenkovic, and Dan Roth. Finding the largest area axis-parallel rectangle in a polygon. *Computational Geometry*, 7(1):125–148, 1997.

9 Paul Fischer and Klaus-Uwe Höffgen. Computing a maximum axis-aligned rectangle in a convex polygon. *Information Processing Letters*, 51(4):189–193, 1994.

10 Partha P. Goswami, Sandip Das, and Subhas C. Nandy. Triangular range counting query in 2D and its application in finding k nearest neighbors of a line segment. *Computational Geometry*, 29(3):163–175, 2004.

11 Olaf Hall-Holt, Matthew J. Katz, Piyush Kumar, Joseph S. B. Mitchell, and Arik Sityon. Finding Large Sticks and Potatoes in Polygons. In *Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA 2016)*, pages 474–483, 2006.

12 Christian Knauer, Lena Schlipf, Jens M. Schmidt, and Hans Raj Tiwary. Largest inscribed rectangles in convex polygons. *Journal of Discrete Algorithms*, 13:78–85, 2012.

13 Michael McKenna, Joseph O'Rourke, and Subhash Suri. Finding the largest rectangle in an orthogonal polygon. In *Proceedings of 23rd Allerton Conference on Communication, Control and Computing*, pages 486–495, 1985.

14 Derick Wood and Chee K. Yap. The orthogonal convex skull problem. *Discrete & Computational Geometry*, 3(4):349–365, 1988.

Motif Counting in Preferential Attachment Graphs

Jan Dreier
Department of Computer Science, RWTH Aachen University, Germany
`https://tcs.rwth-aachen.de/~dreier`
dreier@cs.rwth-aachen.de

Peter Rossmanith
Department of Computer Science, RWTH Aachen University, Germany
`https://tcs.rwth-aachen.de`
rossmani@cs.rwth-aachen.de

Abstract

Network motifs are small patterns that occur in a network significantly more often than expected. They have gathered a lot of interest, as they may describe functional dependencies of complex networks and yield insights into their basic structure [22]. Therefore, a large amount of work went into the development of methods for network motif detection in complex networks [20, 28, 8, 31, 16, 1, 25]. The underlying problem of motif detection is to count how often a copy of a pattern graph H occurs in a target graph G. This problem is #W[1]-hard when parameterized by the size of H [14] and cannot be solved in time $f(|H|)n^{o(|H|)}$ under #ETH [7].

Preferential attachment graphs [3] are a very popular random graph model designed to mimic complex networks. They are constructed by a random process that iteratively adds vertices and attaches them preferentially to vertices that already have high degree. Preferential attachment has been empirically observed in real growing networks [24, 19].

We show that one can count subgraph copies of a graph H in the preferential attachment graph G_m^n (with n vertices and nm edges, where m is usually a small constant) in expected time $f(|H|)m^{O(|H|^6)}\log(n)^{O(|H|^{12})}n$. This means the motif counting problem can be solved in expected quasilinear FPT time on preferential attachment graphs with respect to the parameters $|H|$ and m. In particular, for fixed H and m the expected run time is $O(n^{1+\varepsilon})$ for every $\varepsilon > 0$.

Our results are obtained using new concentration bounds for degrees in preferential attachment graphs. Assume the (total) degree of a set of vertices at a time t of the random process is d. We show that if d is sufficiently large then the degree of the same set at a later time n is likely to be in the interval $(1 \pm \varepsilon)d\sqrt{n/t}$ (for $\varepsilon > 0$) for all $n \geq t$. More specifically, the probability that this interval is left is exponentially small in d.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases random graphs, motif counting, average case analysis, preferential attachment graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.13

1 Introduction

Network motifs are small patterns that occur in a network significantly more often than expected. They are relevant for example in the analysis of biological networks such as transcription networks of bacteria [22]. Detecting network motifs is computationally very expensive and there exist numerous algorithms for this task [20, 28, 8, 31, 16, 1, 25]. The underlying problem of motif detection is to count how often a copy of a pattern graph H occurs in a target graph G. This can be very hard, as counting perfect matchings is #P-hard [29]. One of the fastest algorithms by Curticapean, Dell, and Marx can count subgraph copies of a graph H with k edges in a graph G of size n in time $k^{O(k)}n^{0.174k+o(k)}$ [12]. When it comes to parameterized complexity, counting k-cliques is #W[1]-hard [14] and cannot be done in time $f(k)n^{o(k)}$ under #ETH [7].

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 13; pp. 13:1–13:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A general question is whether problems that are hard on general graphs can be solved efficiently in real-world networks. To this end, the average run time of algorithms on random graphs has been considered (see [15] for a survey from 1997). For example Janson, Łuczak and Norros show that in certain scale-free random graphs with exponent $\alpha > 2$ one can find a maximal clique in polynomial time [18].

Preferential attachment graphs [3] are random graphs designed to mimic complex networks. They are constructed by a random process that iteratively adds vertices and attaches them preferentially to vertices that already have high degree. Scale-free behaviour has been identified as a central property of many complex networks [6, 9] and the preferential attachment process is a widely recognized explanation [5] of this behaviour. Preferential attachment has been empirically observed in real growing networks [24, 19].

Recently, the behaviour of some algorithms on preferential attachment graphs has been analyzed. Let G_m^n be the preferential attachment graph with n vertices and m edges per vertex by (see Section 2 or a rigorous definition). For example, Korula and Lattanzi present a reconciliation algorithm with proven 97% success in preferential attachment graphs [21] and Cooper and Frieze show that the cover time of a simple random walk on G_m^n is with high probability asymptotic to $\frac{2m}{m-1} n \log(n)$ [10].

We show that the motif counting problem can be solved in expected quasilinear time on preferential attachment graphs by a simple algorithm for any motif of constant size. For simple graphs G and H let $\#\mathrm{Sub}(H, G)$ be the number of subgraphs of G isomorphic to H. If the graph G has loops or multi-edges (as preferential attachment graphs do) then the subgraphs is counted with respect to the simple graph corresponding to G. Our main result is the following.

▶ **Theorem 5.** *There exists a function f such that for every graph H and $n, m \in \mathbf{N}$ one can compute $\#\mathrm{Sub}(H, G_m^n)$ in expected time $f(|H|) m^{O(|H|^6)} \log(n)^{O(|H|^{12})} n$.*

This means one can compute $\#\mathrm{Sub}(H, G_m^n)$ in expected quasilinear FPT time on preferential attachment graphs with respect to the parameters $|H|$ and m. In particular, for fixed H and m the expected run time is $O(n^{1+\varepsilon})$ for every $\varepsilon > 0$. Our results can be easily extended to alternative definitions of $\#\mathrm{Sub}(H, G)$ for multigraphs.

Our result is obtained as follows: At first, we define a value $\gamma_l(G)$ for every graph G and $l \in \mathbf{N}^+$ and present a simple algorithm to compute $\#\mathrm{Sub}(H, G)$ in time $f(|H|)\gamma_{|H|}(G)$ for some function f (Lemma 3). We then bound $\gamma_l(G)$ by the number of subgraphs in G of bounded size with at most two pendant vertices (Lemma 7). Using this insight, we can bound the expected value of $\gamma_l(G)$ in preferential attachment graphs by $\mathrm{E}[\gamma_l(G_m^n)] = m^{O(l^6)} \log(n)^{O(l^{12})} n$ (Theorem 4), which directly yields the efficient subgraph counting algorithm. This analysis is based on concentration bounds for vertex degrees, which are proven in Section 5.

Concentration Bounds for Degrees in Preferential Attachment Graphs

A large part of the analysis of our motif counting algorithm is based on concentration bounds for degrees, which we believe to be of individual interest. Aspects of the degree distributions in preferential attachment graphs are well studied [5, 4, 2, 17, 23, 27, 26, 32]. For example, Bollobás et al. [5] show that the degree sequence follows a power law distribution and Peköz et al. [26, 27] bound the rate of convergence of the degree of individual vertices to a limit distribution. The resulting tail bounds for degrees of individual vertices, however, have only polynomial accuracy. We complement these results by providing exponentially strong concentration bounds for vertices or sets of vertices with high degree. We believe these bounds to be useful for proving structural properties and analyzing algorithms on preferential attachment graphs beyond motif counting.

Let the vertices in a preferential attachment graph be $v_1, v_2, v_3, \ldots$ in order of insertion. Let $t \in \mathbf{N}$ and $S \subseteq \{v_1, \ldots, v_t\}$. We analyze the evolution of the degree of S in the random process over time. For $n \geq t$ and $m \geq 1$ we define $d_m^n(S)$ to be the sum over all degrees of vertices in S in G_m^n (we define $d_m^n(v_i) := d_m^n(\{v_i\})$).

Assume the degree of S at a time t to be $d_m^t(S) = d$. It can be shown that the expected degree of S at a later time $n \geq t$ of the same random process asymptotically approaches $\mathrm{E}[d_m^n(S) \mid d_m^t(S) = d] \sim \sqrt{\frac{n}{t}}d$ [30]. In general, the preferential attachment process is too unstable and chaotic to guarantee that the degree of S closely centered around its expected value. We show, however, that if d is sufficiently large then the degree of S at time n is likely to be in the interval $(1 \pm \varepsilon)\sqrt{\frac{n}{t}}d$ (for $\varepsilon > 0$) for all $n \geq t$. More specifically, the probability that this interval is left is exponentially small in d. This is formalized by the following theorem.

▶ **Theorem 19.** *For $t, m, d \in \mathbf{N}^+$, $0 < \varepsilon \leq 1/2$, $S \subseteq \{v_1, \ldots, v_t\}$ with $\Pr[d_m^t(S) = d] \neq 0$ and $d \geq \log(\log(3tm))\varepsilon^{-200}$*

$$\Pr\left[(1-\varepsilon)\sqrt{\frac{n}{t}}d < d_m^n(S) < (1+\varepsilon)\sqrt{\frac{n}{t}}d \text{ for all } n \geq t \;\middle|\; d_m^t(S) = d\right] \geq 1 - e^{-\varepsilon^{200}d}.$$

Note that concentration is guaranteed for all $n \geq t$ simultaneously. This means especially that the degree of large sets of vertices is strongly concentrated at all times of the random process. The constants have been chosen to ease calculations and can be greatly improved.

2 Preliminaries

We will denote probabilities by $\Pr[*]$ and expectation by $\mathrm{E}[*]$. The logarithm is the natural logarithm. We use common graph theory notation [13]. The *order* of a graph is $|G| = |V(G)|$. The *size* of a graph is $\|G\| = |V(G) + E(G)|$. All graphs (except preferential attachment graphs) are simple graphs. The underlying simple graph of a multigraph is obtained by replacing multi-edges with a single edge and removing self-loops. In this work we focus on the *preferential attachment random graph model* [3]. The model generates random graphs by iteratively inserting new vertices and edges. It depends on a parameter m that equals the number of edges attached to a newly created vertex. We follow the definition of Bollobás et al. [5]: For a fixed m, the random process is defined by starting with a single vertex and iteratively adding vertices, thereby constructing a sequence of graphs $G_m^1, G_m^2, \ldots, G_m^t$, where G_m^t has t vertices and mt edges. We define $d_m^t(v)$ to be the degree of vertex v in the graph G_m^t. The random process for $m = 1$ works as follows. A random graph is started with one vertex v_1 that has exactly one self-loop. This graph is G_1^1. We then define the graph process inductively: Given G_1^{t-1} with vertex set $\{v_1, \ldots, v_{t-1}\}$, we create G_1^t by adding a new vertex v_t together with a single edge from v_t to v_i, where i is chosen at random from $\{1, \ldots, t\}$ with

$$\Pr[i = s] = \begin{cases} d_1^{t-1}(v_s)/(2t-1) & 1 \leq s < t, \\ 1/(2t-1) & s = t. \end{cases}$$

This means we add an edge to a random vertex with a probability proportional to its degree at the time. For $m > 1$, the process can be defined by merging sets of m consecutive vertices in G_1^{mt} to single vertices in G_m^t [5]. Let $v_1', \ldots, v_{mt}'$ be the vertices of G_1^{mt}. The graph G_m^t

with vertices $v_1, \dots, v_t$ is constructed by merging $v'_{(i-1)m+1}, \dots, v'_{im}$ into a single vertex v_i. The graph G^t_m is a multigraph. The number of edges between vertices v_i and v_j in G^t_m equals the number of edges between the corresponding sets of vertices in G^{mt}_1. Self-loops and multi-edges are allowed.

In this work we obtain concentration bounds for the total degree of a set of vertices $S \subseteq \{v_1, \dots, v_t\}$ during the random process. We define the degree of a set S at time $n \geq t$ as $d^n_m(S) = \sum_{v \in S} d^n_m(v)$.

3 Subgraph Counting

We start by presenting a very simple algorithm that decides for a graph G and a connected pattern graph H if there exists a subgraph of G isomorphic to H. Then Lemma 2 and 3 generalize this algorithm into a counting algorithm for arbitrary pattern graphs.

Assume there is a subgraph H' of G isomorphic to H that we want to find and let $l = |H|$. Since H' is a connected graph with at most l vertices there exists a vertex $v \in V(G)$ such that H' is contained in the l-neighborhood $G[N^G_l(v)]$ of v. We build a spanning tree T of $G[N^G_l(v)]$. Since T is a tree, it is fairly easy to find H' if H' is a subgraph of T. But what happens if H' contains edges that are not contained in T? We call the edges of $G[N^G_l(v)]$ that are not in T the *extra edges* of the l-neighborhood of v. Since H has at most $\binom{l}{2}$ edges, there exists a subset F of at most $\binom{l}{2}$ many extra edges such that H is contained in $(V(T), E(T) \cup F)$. The graph $(V(T), E(T) \cup F)$ is a tree with at most $\binom{l}{2}$ extra edges and therefore has bounded treewidth. Using Courcelle's theorem [11] it is still easy to find H' in $(V(T), E(T) \cup F)$. In summary, one can find the graph H' by enumerating all $v \in V(G)$ and sets F of at most $\binom{l}{2}$ extra edges in the l-neighborhood of v in G, and then using Courcelle's theorem.

We define a value $\gamma_l(G)$ of a graph G, which can be obtained by multiplying the size of each l-neighborhood with the number of sets of extra edges of size at most $\binom{l}{2}$.

▶ **Definition 1.** *Let G be a graph and $l \in \mathbf{N}^+$. We define*

$$\gamma_l(G) = \sum_{v \in V(G)} |N^G_l(v)| \sum_{k=0}^{\binom{l}{2}} \binom{\|G[N^G_l(v)]\| - |N^G_l(v)| - 1}{k}.$$

For multigraphs G, $\gamma_l(G)$ is defined with respect to the simple underlying graph.

We now show that $\gamma_l(G)$ captures the run time of the previously discussed algorithm (up to a factor independent of G). We start with counting connected patterns and generalize this afterwards to arbitrary patterns.

▶ **Lemma 2.** *There exists a function f such that for every graph G and connected graph H one can compute $\#\mathrm{Sub}(H, G)$ in time $f(|H|)\gamma_{|H|}(G)$.*

Proof. Let $l = |H|$. We compute spanning trees T_v of $G[N^G_l(v)]$ for $v \in V(G)$ in time $\sum_{v \in V(G)} O(\|G[N^G_l(v)]\|)$ by breadth-first searches. Let $\mathcal{F}_v := \{F \mid F \subseteq E(G[N^G_l(v)]) \setminus E(T_v), |F| \leq \binom{l}{2}\}$ be the set of all subsets of at most $\binom{l}{2}$ edges that are in $G[N^G_l(v)]$ but not in T_v. We construct the sets $\mathcal{F}_v$ for $v \in V(G)$ in time $\sum_{v \in V(G)} O(\|G[N^G_l(v)]\| + |\mathcal{F}_v| l^2)$.

Let I be the set of all subgraphs of G isomorphic to H. For $v \in V(G)$ and $F \in \mathcal{F}_v$ let $I_{v,F}$ be the set of subgraphs H' of $(V(T_v), E(T_v) \cup F)$ such that H' is isomorphic to H, $v \in V(H')$ and $F \subseteq E(H')$. We claim that $\#\mathrm{Sub}(H, G) = |I| = \sum_{v \in V} \sum_{F \in \mathcal{F}_v} \frac{|I_{v,F}(H)|}{|H|}$. Let H' be a subgraph of G. If H' is not isomorphic to H then by definition $H' \notin I_{v,F}$ for all

$v \in V(G)$, $F \in \mathcal{F}_v$. Assume now that H' is isomorphic to H. To prove the claim, need to make sure that H' is counted exactly $|H|$ times. This is the case because $H' \in I_{v,F}$ if and only if $v \in V(H')$ and $F = E(H') \setminus E(T_v)$.

In order to compute $\#\mathrm{Sub}(H, G)$, it is now sufficient to iterate over all $v \in V$ and $F \in \mathcal{F}_v$ and compute $|I_{v,F}|$. The graph $(V(T_v), E(T_v) \cup F)$ is a tree with at most $\binom{l}{2}$ additional edges and therefore has treewidth at most $\binom{l}{2} + 1$. By Courcelle's theorem [11], there exists a function f' such that one can compute $|I_{v,F}|$ in time $f'(l)|N_l^G(v)|$.

The run time of this procedure is dominated by the time taken to compute $T_v, \mathcal{F}_v$ for $v \in V(G)$ and $|I_{v,F}|$ for $v \in V, F \in \mathcal{F}_v$. Since $\|G[N_l^G(v)]\| \leq |N_l^G(v)| + |\mathcal{F}_v|$, this run time is bounded by

$$\sum_{v \in V(G)} O\Big(\|G[N_l^G(v)]\| + |\mathcal{F}_v| l^2 + |\mathcal{F}_v| f'(l) |N_l^G(v)|\Big) = O\big(f'(l) \sum_{v \in V} |\mathcal{F}_v||N_l^G(v)|\big). \quad ◀$$

▶ **Lemma 3.** *There exists a function f such that for graphs G and H one can compute $\#\mathrm{Sub}(H, G)$ in time $f(|H|)\gamma_{|H|}(G)$.*

Proof. (Sketch) Let $\mathcal{H}$ be a representative set of all connected pairwise non-isomorphic graphs with at most $|H|$ vertices. We compute $\#\mathrm{Sub}(H', G)$ for every connected graph $H' \in \mathcal{H}$. Via inclusion-exclusion, we can compute $\#\mathrm{Sub}(H, G)$. We sketch how the procedure works if H consists of two components. Via induction, it can be generalized to an arbitrary number of components. Let C_1 and C_2 be the components of H. The value $c = \#\mathrm{Sub}(C_1, G) \cdot \#\mathrm{Sub}(C_2, G)$ counts all ways in which the two components of H can be embedded in G. However, c might be larger than $\#\mathrm{Sub}(H, G)$ since it also counts all embeddings where the two components intersect in G by sharing one or more vertices. Every intersection of the two components is connected, thus, we can count them and subtract them. ◀

We now have a subgraph counting algorithm with efficient run time if the function $\gamma_{|H|}(G)$ is small. If G has bounded degree or is a tree, then $\gamma_{|H|}(G)$ is an fpt function for the parameter $|H|$. It remains to show that the function is also small for certain random graphs.

4 Bounding γ_l in Preferential Attachment Graphs

The remainder of this paper is concerned with the analysis of the run time of the aforementioned algorithm on preferential attachment graphs. This is done by using our concentration bounds for degrees (Theorem 19) to prove the following theorem.

▶ **Theorem 4.** *Let $l, n, m \in \mathbf{N}^+$ with $n \geq 2$. Then $\mathrm{E}[\gamma_l(G_m^n)] = m^{O(l^6)} \log(n)^{O(l^{12})} n$.*

This is then sufficient to prove our main result.

▶ **Theorem 5.** *There exists a function f such that for every graph H and $n, m \in \mathbf{N}$ one can compute $\#\mathrm{Sub}(H, G_m^n)$ in expected time $f(|H|) m^{O(|H|^6)} \log(n)^{O(|H|^{12})} n$.*

Proof. Direct consequence of Lemma 3 and Theorem 4. ◀

We prove Theorem 4 via multiple steps. In Lemma 7, we bound for every graph G and $l \in \mathbf{N}^+$, $\gamma_l(G) \leq 16 l^6 \, |B_{4l^3}^2(G)|$, where $B_l^b(G)$ is defined below.

▶ **Definition 6.** *For a graph G and $l, b \in \mathbf{N}$ let $B_l^b(G)$ be the set of subgraphs in G of size at most l with no isolated vertices and at most b pendant vertices. If G is a multigraph then $B_l^b(G)$ is defined with respect to the simple underlying graph.*

Then we use the degree bounds from Theorem 19 to step by step (Lemma 8 – 11) bound the expected value of $|B^b_l(G)|$ in preferential attachment graphs.

▶ **Lemma 7.** *Let G be a graph and $l \in \mathbf{N}^+$. Then $\gamma_l(G) \leq 16l^6\,|B^2_{4l^3}(G)|$.*

Proof. For every $v \in V(G)$ let T_v be a breadth-first spanning tree with root v in $G[N^G_l(v)]$ and $\mathcal{F}_v := \{F \mid F \subseteq E(G[N^G_l(v)]) \setminus E(T_v), |F| \leq \binom{l}{2}\}$ be the set of all subsets of at most $\binom{l}{2}$ edges that are in $G[N^G_l(v)]$ but not in T_v. Clearly $\gamma_l(G) = \sum_{v \in V(G)} |N^G_l(v)||\mathcal{F}_v|$.

Let $v \in V(G)$, $w \in N^G_l(v)$, $F \in \mathcal{F}_v$. Let $U \subseteq V(G)$ be the set containing v, w and all endpoints of the edges in F. We define a graph $H_{v,w,F}$ as follows: Start with the empty graph, add the vertices U, the edges F, and for every $u \in U$ the unique path in T_v from v to u. Since T_v is a breadth-first spanning tree, every path in T_v starting at v contains at most $l+1$ vertices. Since also $|U| \leq 2\binom{l}{2} + 2$, we can bound $V(H_{v,w,F}) \leq (2\binom{l}{2} + 2)(l+1) \leq 4l^3$. Furthermore $H_{v,w,F}$ contains no vertices with degree zero and every vertex in $H_{v,w,F}$ except for v and w is guaranteed to have degree at least two. This implies $H_{v,w,F} \in B^2_{4l^3}(G)$.

Let further $v' \in V(G)$, $w' \in N^G_l(v)$. If there exists $F' \in \mathcal{F}(v')$ with $H_{v,w,F} = H_{v',w',F'}$ then $v \in V(H_{v,w,s})$ and $w \in V(H_{v,w,s})$. Also there exists at most one $F' \in \mathcal{F}_v$ such that $H_{v,w,F} = H_{v',w',F'}$. Thus, there are at most $16l^6$ choices for v', w', F' such that $H_{v,w,F} = H_{v',w',F'}$. ◀

It is now sufficient to bound the expected value of $|B^b_l(G)|$ in preferential attachment graphs. At first, we use Theorem 19 to give an upper bound on the degrees of single vertices.

▶ **Lemma 8.** *There exists $h > 0$ such that for $a \in \mathbf{R}$, $n, t, d \in \mathbf{N}^+$ with $n \geq at$, and $a \geq h \log\log(3at)$ it holds that $\Pr\left[d^n_1(v_t) \geq a\sqrt{\frac{n}{t}}\right] \leq e^{-a/h}$.*

Proof. Let $S = \{v_t, \ldots, v_{t+\lceil a/5\rceil}\}$. Then $d^n_1(v_t) \leq d^n_1(S)$. We assume h to be large enough that $a \geq 1000$. Therefore $t + \lceil a/5\rceil \leq at \leq n$ and $a/5 \leq d^{t+\lceil a/5\rceil}_1(S) \leq 2\lceil 1 + a/5\rceil \leq a/2$. We use these inequalities to bound

$$\Pr\left[d^n_1(v_t) \geq a\sqrt{\frac{n}{t}}\right] \leq \sum_{d=\lceil a/5\rceil}^{\lfloor a/2\rfloor} \Pr\left[d^{t+\lceil a/5\rceil}_1(S) = d\right] \Pr\left[d^n_1(S) \geq 2\sqrt{\frac{n}{t}}d \;\middle|\; d^{t+\lceil a/5\rceil}_1(S) = d\right].$$

Let $\varepsilon = 1/2$. We choose h large enough such that $a/5 \geq \log(\log(3(t + \lceil a/5\rceil)))\varepsilon^{-200}$ and $\varepsilon^{200}a/5 \geq a/h$. Theorem 19 yields for $\lceil a/5\rceil \leq d \leq \lfloor a/2\rfloor$

$$\Pr\left[d^n_1(S) \geq (1+\varepsilon)\sqrt{\frac{n}{t + \lceil a/5\rceil}}d \;\middle|\; d^{t+\lceil a/5\rceil}_1(S) = d\right] \leq e^{-\varepsilon^{200}d} \leq e^{-a/h}. \quad ◀$$

While it is easy to use the expected degree of a vertex to show that the probability that a single edge v_xv_y exists in G^n_1 is close to $1/\sqrt{xy}$, it is surprisingly involved to bound the probability that multiple edges occur. This is because the existence of some edges influences the degree. Lemma 8 helps us here. We first show the result for $m = 1$ (Lemma 9) and then lift it to arbitrary values of m (Lemma 10).

▶ **Lemma 9.** *Let $n \geq 2$ and $E \subseteq \binom{\{v_1,\ldots,v_n\}}{2}$. Then*

$$\Pr[E \subseteq E(G^n_1)] \leq \log(n)^{O(|E|)^2} \prod_{v_xv_y \in E} 1/\sqrt{xy}.$$

Proof. We can assume E that $E = \{v_{x_1}v_{y_1}, \dots, v_{x_l}v_{y_l}\}$ with $x_i < y_i$ for $1 \le i \le l$ and $y_i < y_j$ if $i < j$. Also, we define for $k \le l$, $E_k = \{v_{x_1}v_{y_1}, \dots, v_{x_k}v_{y_k}\}$ as the subset of the first k edges. The chain rule gives us

$$\Pr[E \subseteq E(G_1^n)] = \prod_{k=1}^{l} \Pr[v_{x_k}v_{y_k} \in E(G_1^n) \mid E_{k-1} \subseteq E(G_1^n)].$$

We fix some $1 \le k \le l$ and set $x = x_k$, $y = y_k$. It is now sufficient to show that

$$\Pr[v_x v_y \in E(G_1^n) \mid E_{k-1} \subseteq E(G_1^n)] \le \log(n)^{O(k)}/\sqrt{xy}.$$

If $d_1^{y-1}(v_x) = l$ for $l \in \mathbf{N}$ then the edge $v_x v_y$ is inserted with probability $l/(2y-1)$. Thus

$$\begin{aligned}\Pr[v_x v_y \in E(G_1^n) \mid E_{k-1} \subseteq E(G_1^n)] &= \sum_{l=1}^{\infty} l/(2y-1) \cdot \Pr[d_1^{y-1}(v_x) = l \mid E_{k-1} \subseteq E(G_1^n)] \\ &= 1/(2y-1) \cdot \mathrm{E}[d_1^{y-1}(v_x) \mid E_{k-1} \subseteq E(G_1^n)] \le \mathrm{E}[d_1^{y}(v_x) \mid E_{k-1} \subseteq E(G_1^n)]/y. \quad (1)\end{aligned}$$

Let now $\lambda \in \mathbf{R}$, whose value we will specify later. Since $d_1^y(v_x) \le 2y$, the law of total probability states

$$\begin{aligned}\mathrm{E}[d_1^y(v_x) \mid E_{k-1} \subseteq E(G_1^n)] &\le \lambda + 2y \Pr[d_1^y(v_x) > \lambda \mid E_{k-1} \subseteq E(G_1^n)] \\ &\le \lambda + 2y \Pr[d_1^y(v_x) > \lambda] / \Pr[E_{k-1} \subseteq E(G_1^n)]. \quad (2)\end{aligned}$$

We now need to find a lower bound for $\Pr[E_{k-1} \subseteq E(G_1^n)]$. For the first y steps the summed degree of all vertices is at most $2y$. Also each vertex has degree at least one. This means that every individual edge has probability at least $1/2y$, independent of where previous edges are. This observation together with the chain rule yields

$$\Pr[E_{k-1} \subseteq E(G_1^n)] = \prod_{i=1}^{k-1} \Pr[v_{x_i}v_{y_i} \in E(G_1^n) \mid E_{i-1} \subseteq E(G_1^n)] \le 1/(2y)^k. \quad (3)$$

Combining (1), (2), and (3) yields

$$\Pr[v_x v_y \in E(G_1^n) \mid E \subseteq E(G_1^n)] \le \lambda/y + 2y \Pr[d_1^y(v_x) > \lambda](2y)^k/y. \quad (4)$$

Let h be the constant from Lemma 8. We now set $\lambda = h\log(y)^{2k}\sqrt{y/x}$. Then (4) and Lemma 8 (with $a = h\log(y)^{2k}$ and $e^{-a/h} = y^{-2k}$) yield

$$\Pr[v_x v_y \in E(G_1^n) \mid E \subseteq E(G_1^n)] \le h\log(y)^{2k}\sqrt{y/x}/y + 2y^{-2k}(2y)^k = \log(n)^{O(k)}/\sqrt{xy}. \blacktriangleleft$$

▶ **Lemma 10.** *Let $n, m \in \mathbf{N}^+$, $n \ge 2$ and $E \subseteq \binom{\{v_1,\dots,v_n\}}{2}$. Then*

$$\Pr[E \subseteq E(G_m^n)] \le \log(n)^{O(|E|)^2} m^{2|E|} \prod_{v_x v_y \in E} 1/\sqrt{xy}.$$

Proof. One can simulate G_m^n via G_1^{mn}, by merging every m consecutive vertices into a single one. For $v_x v_y \in E$ let $E_{xy} = \{v_{x'}v_{y'} \mid m(x-1)+1 \le x' \le mx, m(y-1)+1 \le y' \le my\}$. This means the edge $v_x v_y$ is present after the merge operation in G_m^n if any edge from E_{xy} is present in G_1^{mn}. The union bound and Lemma 9 yield

$$\begin{aligned}\Pr[E \subseteq E(G_m^n)] \le \log(n)^{O(|E|)^2} \prod_{v_x v_y \in E} \sum_{v_{x'}v_{y'} \in E_{xy}} 1/\sqrt{x'y'} \\ \le \log(n)^{O(|E|)^2} m^{2|E|} \prod_{v_x v_y \in E} 1/\sqrt{xy}.\end{aligned}$$

◀

We can now bound $\mathrm{E}[|B_l^b(G_m^n)|]$ by iterating over all possible embeddings of graphs of size at most l with no isolated vertices and b pendant vertices into G_m^n. We use Lemma 10 to bound the probability that the edges required for this embedding are indeed present in G_m^n.

▶ **Lemma 11.** *Let $l, b, n, m \in \mathbf{N}^+$ with $n \geq 2$. Then $\mathrm{E}[|B_l^b(G_m^n)|] = n^{b/2}\log(n)^{O(l^4)}m^{O(l^2)}$.*

Proof. Let H be a graph with at most l vertices, at most b pendant vertices and no isolated vertices. Let p be the expected number of subgraphs of G_m^n that are isomorphic to H. We want to give an upper bound for p. Let $V(H) = \{u_1, \dots, u_\gamma\}$ with $\gamma \leq l$ and let $\delta_1, \dots, \delta_\gamma$ be the degree sequence of $V(H)$. We compute the following bound for later

$$\sum_{x_i=1}^{n} \frac{1}{\sqrt{x_i^{\delta_i}}} \leq 1 + \int_1^n \frac{1}{\sqrt{x^{\delta_i}}} dx \leq 1 + \begin{cases} \log(n) & \text{if } \delta_i \geq 2, \\ 2\sqrt{n} & \text{if } \delta_i = 1. \end{cases} \tag{5}$$

For integers $1 \leq x_1, \dots, x_\gamma \leq n$, we consider an embedding of H into G_m^n that maps u_i to v_{x_i} (for $1 \leq i \leq \gamma$). According to Lemma 10, the probability that this embedding of H is a subgraph of G_m^n is at most $\log(n)^{O(l^4)}m^{O(l^2)}\prod_{i=1}^{\gamma}\frac{1}{\sqrt{x_i}^{\delta_i}}$. We sum over all possible embeddings and use (5) to bound p by

$$\sum_{x_1=1}^{n} \cdots \sum_{x_\gamma=1}^{n} \log(n)^{O(l^4)}m^{O(l^2)} \prod_{i=1}^{\gamma} \frac{1}{\sqrt{x_i^{\delta_i}}} = \log(n)^{O(l^4)}m^{O(l^2)} \sum_{x_1=1}^{n} \frac{1}{\sqrt{x_1^{\delta_1}}} \cdots \sum_{x_\gamma=1}^{n} \frac{1}{\sqrt{x_\gamma^{\delta_\gamma}}}$$
$$\overset{(5)}{=} \log(n)^{O(l^4)}m^{O(l^2)}(1+\log(n))^\gamma(1+2\sqrt{n})^b = n^{b/2}\log(n)^{O(l^4)}m^{O(l^2)}.$$

For an arbitrary but fixed graph H with at most l vertices, no isolated vertices and at most b pendant vertices we have bound the expected number of occurrences p. There are no more than 2^{l^2} graphs with at most l fixed vertices. Therefore, $\mathrm{E}[|B_l^b(G_m^n)|] \leq 2^{l^2}n^{b/2}\log(n)^{O(l^4)}m^{O(l^2)}$. ◀

At last, Theorem 4 is a direct consequence Lemma 7 and Lemma 11.

5 Degree Bounds

In this section we show that under certain conditions the degree of vertices is closely centered around their expected value. This is formalized in Theorem 19, which is proven at the end of this section. We separately show upper and lower bounds and then join these bounds together. These bounds are proven by first giving bounds that hold for a short interval of time (Section 5.1) and then extending these bounds for longer intervals of time (Section 5.2).

Let $n \geq t$ and $S \subseteq \{v_1, \dots, v_t\}$. Remember that $d_m^n(S)$ is the degree of a set S in G_m^n. Due to the technical nature of this section, we sometimes consider the set $S \subseteq \{v_1, \dots, v_t\}$ to be fixed and write $D(n)$ as shorthand for $d_1^n(S)$ to avoid having large formulas as a superscript. We also define $D(n) := D(\lfloor n \rfloor)$ for $n \in \mathbf{R}$. For $n > t$ we can explicitly state the probability distribution of $D(n)$ under the condition $D(n-1)$ as

$$\Pr[D(n) = x \mid D(n-1)] = \begin{cases} D(n-1)/(2n-1) & x = D(n-1)+1 \\ 1 - D(n-1)/(2n-1) & x = D(n-1) \\ 0 & \text{otherwise.} \end{cases}$$

5.1 Short-Term Degree Bounds

Here we show that for small δ from time-step t to $(1+\delta)t$ it is very likely that we increase the degree of the set S by a factor of $1+\delta/2+O(\delta^2)$.

▶ **Lemma 12.** *Let $0<\delta<1$ and $t \geq \frac{2}{\delta^2}$. Then*

$$\Pr\Big[D\big((1+\delta)t\big) \leq \big(1+\frac{\delta}{2}-2\delta^2\big)D(t) \mid D(t)\Big] \leq e^{-\frac{1}{16}\delta^3 D(t)}.$$

Proof. For every $t' \in \mathbf{R}$ $D(t')=D(\lfloor t'\rfloor)$. For every $t' \in \mathbf{N}$ either $D(t')=D(t'-1)$ or $D(t')=D(t'-1)+1$. Let N be the number of integers between t and $(1+\delta)t$. Let Δ_i with $1 \leq i \leq N$ be the Bernoulli variable indicating that $D(\lfloor t\rfloor+i)=D(\lfloor t\rfloor+i-1)+1$ and $\Delta=\Delta_1+\cdots+\Delta_N$. Then $D(t)+\Delta=D((1+\delta)t)$. Furthermore

$$\Pr[\Delta_i=1 \mid \Delta_1,\ldots,\Delta_{i-1},D(t)]=\frac{D(\lfloor t\rfloor+i-1)}{2(\lfloor t\rfloor+i)-1} \geq \frac{D(t)}{2(1+\delta)t}.$$

Let $X=X_1+\cdots+X_N$ be the sum of identically distributed Bernoulli variables with $\Pr[X_i=1]=\frac{D(t)}{2(1+\delta)t}$. We consider two experiments: The first game is N tosses of a fair coin. The second one is N tosses of a biased coin, where the probability that the ith coin comes up head depends on the outcome of the previous coins but always is at least $1/2$. Obviously, the probability of at least s heads in the second experiment is at least as high as the probability of at least s heads in the first experiment. The same argument implies

$$\Pr[\Delta \leq s \mid D(t)] \leq \Pr[X \leq s \mid D(t)]. \tag{6}$$

With $t \geq \frac{2}{\delta^2}$ we get $N \geq \delta t-1 \geq (\delta-\frac{1}{2}\delta^2)t$ and

$$E[X \mid D(t)]=N\Pr[X_i=1 \mid D(t)] \geq \frac{(\delta-\delta^2/2)D(t)}{2(1+\delta)}. \tag{7}$$

In contrast to Δ, we can directly apply Chernoff bounds to X:

$$\Pr\Big[X \leq (1-\delta)E\big[X \mid D(t)\big] \;\Big|\; D(t)\Big] \leq e^{-\frac{1}{2}\delta^2 E[X|D(t)]}. \tag{8}$$

Combining the above inequality with (7), (6) and (8) yields

$$\Pr\Big[\Delta \leq \frac{(1-\delta)(\delta-\delta^2/2)D(t)}{2(1+\delta)} \;\Big|\; D(t)\Big] \overset{(6)(7)}{\leq} \Pr\Big[X \leq (1-\delta)E[X \mid D(t)] \;\Big|\; D(t)\Big]$$
$$\overset{(8)}{\leq} e^{-\frac{1}{2}\delta^2 E[X|D(t)]} \overset{(7)}{\leq} e^{-\frac{\delta^3-\delta^4/2}{4(1+\delta)}D(t)} \leq e^{-\frac{1}{16}\delta^3 D(t)}. \tag{9}$$

For $0 \leq \delta \leq 1$, $\frac{(1-\delta)(\delta-\delta^2/2)}{2(1+\delta)} \geq \frac{\delta}{2}-2\delta^2$. Thus, by (9) and $D((1+\delta)t)=\Delta+D(t)$

$$\Pr\big[D((1+\delta)t) \leq (1+\delta/2-2\delta^2)D(t) \bigm| D(t)\big]=\Pr\big[\Delta \leq (\delta/2-2\delta^2)D(t) \bigm| D(t)\big]$$
$$\leq e^{-\frac{1}{16}\delta^3 D(t)}.$$
◀

Unfortunately, an additional factor of $\log(2et)$ is introduced in the following upper bound. The proof is very similar to the previous one and is omitted for lack of space.

▶ **Lemma 13.** *Let $0<\delta \leq \frac{1}{e^2}$ and $t \geq \frac{2}{\delta^2}$. Then*

$$\Pr\Big[D((1+\delta)t) \geq (1+\delta/2+2\delta^2)D(t) \;\Big|\; D(t)\Big] \leq \log(2et)e^{-\frac{1}{8}\delta^3 D(t)}.$$

5.2 Long-Term Degree Bounds

In the previous subsection we established bounds for a small interval from step t to step $(1+\delta)t$ with an error of order δ^2. In this subsection we combine these bounds into long-term bounds. We get these bounds by defining positions $t_0 = t$ and $t_{k+1} = (1+\delta_k)t_k$ with $k \in \mathbf{N}$ and using the union bound to guarantee that for each interval from time t_k to t_{k+1} the short-term bounds hold. The choice of δ_k is of high importance for the success of this strategy. It turns out that we need the product $\prod_{k=1}^{\infty}(1+\delta_k)$ to diverge, but the error $\prod_{k=1}^{\infty}(1+\delta_k^2)$ to converge. We settle for $\delta_k = \varepsilon/k^{2/3}$, which satisfies both conditions.

Lemma 14 and Lemma 15 bridge the gap between the bounds for small intervals and longer periods by stating that if the degree differs by a factor of $(1 \pm \varepsilon)$ from its expected value then there has been one interval where the allowed error $O(\delta^2)$ has been exceeded.

▶ **Lemma 14.** *Let $0 < \varepsilon \leq 1/8$, $t > 0$, and $f \colon \mathbf{R} \to \mathbf{R}$ be an increasing function. For every $k \in \mathbf{N}$ let $\delta_k = \frac{\varepsilon}{k^{2/3}}$, $h_k = \prod_{i=1}^{k-1}(1+\delta_i)$, and $c_k = \prod_{i=1}^{k-1}(1+\frac{1}{2}\delta_i - 2\delta_i^2)$.*

If there is an $n \in \mathbf{N}$, such that $t < n$ and $f(n) < (1-\varepsilon)\sqrt{\frac{n}{t}}f(t)$, then there is a $k \in \mathbf{N}$ such that $f((1+\delta_k)h_k t) < (1+\frac{1}{2}\delta_k - 2\delta_k^2)f(h_k t)$ and $f(h_k t) \geq c_k f(t)$.

Proof. Consider any $n \in \mathbf{N}$, $n \geq t$. Let $k(n) \in \mathbf{N}$ be the maximal value such that $h_{k(n)}t \leq n$. Then $\frac{n}{1+\delta_{k(n)}} \leq h_{k(n)}t$, because of the maximality of $k(n)$. Notice that

$$(1-\varepsilon)\sqrt{\frac{n}{t}} \leq e^{-\frac{1}{2}\varepsilon}e^{-\frac{1}{2}\varepsilon}\sqrt{\frac{n}{t}} = e^{-\frac{1}{2}\varepsilon}\sqrt{\frac{n}{te^{\varepsilon}}} \leq e^{-\frac{1}{2}\varepsilon}\sqrt{\frac{n}{t(1+\delta_{k(n)})}} \leq e^{-\frac{1}{2}\varepsilon}\sqrt{h_{k(n)}}$$

and for all $k \in \mathbf{N}$

$$c_k \geq \prod_{i=1}^{k-1} e^{\frac{1}{2}\delta_i - 3\delta_i^2} \geq \Big(\prod_{i=1}^{k-1} e^{\delta_i}\Big)^{\frac{1}{2}} \prod_{i=1}^{\infty} e^{-\frac{3\varepsilon^2}{i^{4/3}}} \geq \Big(\prod_{i=1}^{k-1}(1+\delta_i)\Big)^{\frac{1}{2}} e^{-4\varepsilon^2} \geq e^{-\frac{1}{2}\varepsilon}\sqrt{h_k}.$$

Combining the upper two inequalities gives us $(1-\varepsilon)\sqrt{n/t} \leq c_k$. We assumed $f(n) < (1-\varepsilon)\sqrt{\frac{n}{t}}f(t)$. Monotonicity of f yields $f(h_{k(n)}t) \leq f(n) < (1-\varepsilon)\sqrt{\frac{n}{t}}f(t) \leq c_{k(n)}f(t)$.

Let $J = \{\, j \geq 0 \mid f(h_{j+1}t) < c_{j+1}f(t) \,\}$. The set J is not empty because $k(n) - 1 \in J$ by the equation above. Furthermore, $0 \notin J$ because $h_1 = c_1 = 1$ and therefore $f(h_1 t) = f(t) = c_1 f(t)$. Let now k be the minimal value in J. Then $k > 0$, $f(h_k t) \geq c_k f(t)$, and $f(h_{k+1}t) < c_{k+1}f(t)$. At last, we have

$$f((1+\delta_k)h_k t) = f(h_{k+1}t) < c_{k+1}f(t) = (1+\frac{1}{2}\delta_k - 2\delta_k^2)c_k f(t) \leq (1+\frac{1}{2}\delta_k - 2\delta_k^2)f(h_k t). \quad ◀$$

The proof of Lemma 15 is similar to the one of Lemma 14 and is therefore omitted.

▶ **Lemma 15.** *Let $0 < \varepsilon \leq 1/40$, $t > 0$, and $f \colon \mathbf{R} \to \mathbf{R}$ be an increasing function. For every $k \in \mathbf{N}$ let $\delta_k = \frac{\varepsilon}{k^{2/3}}$, $h_k = \prod_{i=1}^{k-1}(1+\delta_i)$, and $c_k = \prod_{i=1}^{k-1}(1+\frac{1}{2}\delta_i + 2\delta_i^2)$.*

If there is an $n \in \mathbf{N}$, such that $t < n$ and $f(n) > (1+\varepsilon)\sqrt{\frac{n}{t}}f(t)$, then there is a $k \in \mathbf{N}$ such that $f((1+\delta_k)h_k t) > (1+\frac{1}{2}\delta_k + 2\delta_k^2)f(h_k t)$ and $f(h_k t) \leq c_k f(t)$.

▶ **Lemma 16.** *Let $0 < \varepsilon \leq 1/40$, $t > \frac{1}{\varepsilon^6}$. For every $k \in \mathbf{N}$ let $\delta_k = \frac{\varepsilon}{k^{2/3}}$, $h_k = \prod_{i=1}^{k-1}(1+\delta_i)$, $c_k^+ = \prod_{i=1}^{k-1}(1+\frac{1}{2}\delta_i + 2\delta_i^2)$ and $c_k^- = \prod_{i=1}^{k-1}(1+\frac{1}{2}\delta_i - 2\delta_i^2)$. Then*

$$\Pr\Big[D((1+\delta_k)h_k t) < (1+\frac{1}{2}\delta_k - 2\delta_k^2)D(h_k t), D(h_k t) \geq c_k^- D(t) \;\Big|\; D(t)\Big] \leq e^{-\frac{1}{16}\delta_k^3 c_k^- D(t)},$$

$$\Pr\Big[D((1+\delta_k)h_k t) > (1+\frac{1}{2}\delta_k + 2\delta_k^2)D(h_k t), D(h_k t) \leq c_k^+ D(t) \;\Big|\; D(t)\Big]$$
$$\leq \log(2et)e^{-\frac{1}{8}\delta_k^3 c_k^+ D(t)}.$$

Proof. At first we focus on the first bound. By the law of total probability

$$\begin{aligned}\Pr\Big[D((1+\delta_k)h_kt) < (1+\frac{1}{2}\delta_k - 2\delta_k^2)D(h_kt), D(h_kt) \geq c_k^- D(t) \bigm| D(t)\Big]\\ \leq \Pr\Big[D((1+\delta_k)h_kt) < (1+\frac{1}{2}\delta_k - 2\delta_k^2)D(h_kt) \bigm| D(h_kt) \geq c_k^- D(t)\Big].\end{aligned}$$

The second line of this equation states the probability that the degree of a vertex is in the future below a certain threshold under the condition that it is currently above a certain threshold. We can bound this probability if we assume that it currently is not above, but exactly at the threshold:

$$\begin{aligned}\Pr\Big[D((1+\delta_k)h_kt) < (1+\frac{1}{2}\delta_k - 2\delta_k^2)D(h_kt) \bigm| D(h_kt) \geq c_k^- D(t)\Big]\\ \leq \Pr\Big[D((1+\delta_k)h_kt) < (1+\frac{1}{2}\delta_k - 2\delta_k^2)D(h_kt) \bigm| D(h_kt) = c_k^- D(t)\Big].\end{aligned}$$

Similarly, the probability that the degree of a vertex is in the future above a certain threshold under the condition that it is currently below a certain threshold can be bounded by assuming that it is exactly at the threshold. Thus, it suffices to prove the following two bounds

$$\Pr\Big[D((1+\delta_k)h_kt) < (1+\frac{1}{2}\delta_k - 2\delta_k^2)D(h_kt) \bigm| D(h_kt) = c_k^- D(t)\Big] \leq e^{-\frac{1}{16}\delta_k^3 c_k^- D(t)},$$

$$\Pr\Big[D((1+\delta_k)h_kt) > (1+\frac{1}{2}\delta_k + 2\delta_k^2)D(h_kt) \bigm| D(h_kt) = c_k^+ D(t)\Big] \leq \log(2et)e^{-\frac{1}{8}\delta_k^3 c_k^+ D(t)}.$$

Lemma 12 and 13 state that if $0 \leq \delta_k = e/k^{3/2} \leq 1/e^2$ and $h_kt \geq 2/\delta_k^2$ for every k then these bounds are true. We observe that for $0 \leq \varepsilon \leq 1/8$ the first precondition is always satisfied. We will finish the proof by showing that $h_kt \geq 2/\delta_k^2$ for every k. Observe that for $0 \leq k \leq 1$ we have $h_kt \geq 2/\varepsilon^2 \geq 2/\delta_k^2$. We can therefore assume $k \geq 2$. First, we need a lower bound for h_k.

$$h_k = \prod_{i=1}^{k-1}(1+\frac{\varepsilon}{i^{2/3}}) \geq \prod_{i=1}^{k-1} e^{\frac{\varepsilon}{i^{2/3}}} \geq e^{3\varepsilon(k-1)^{1/3}} \geq e^{2\varepsilon k^{1/3}}$$

One can show that $e^x/x^4 \geq e^4/256$ for $x > 0$. We therefore get for $x = 2\varepsilon k^{1/3}$

$$h_k \geq e^{2\varepsilon k^{1/3}} = \frac{e^{2\varepsilon k^{1/3}}}{(2\varepsilon k^{1/3})^4}\frac{16\varepsilon^6}{\delta_k^2} \geq \frac{e^x}{x^4}\frac{16\varepsilon^6}{\delta_k^2} \geq \frac{e^4}{256}\frac{16\varepsilon^6}{\delta_k^2} \geq \frac{2\varepsilon^6}{\delta_k^2}.$$

Since $t \geq \frac{1}{\varepsilon^6}$ it follows that $h_kt \geq \frac{2}{\delta_k^2}$. ◀

▶ **Lemma 17.** *For* $0 < \varepsilon \leq 1/40$ *and* $\frac{1}{\varepsilon^6} < t \in \mathbf{N}$

$$\begin{aligned}\Pr\Big[(1-\varepsilon)\sqrt{\frac{n}{t}}D(t) < D(n) < (1+\varepsilon)\sqrt{\frac{n}{t}}D(t) \text{ for all } n \geq t \Bigm| D(t)\Big]\\ \geq 1 - \log(15t)\varepsilon^{-6}\exp\big(-\varepsilon^{15}10^{-24}D(t)\big).\end{aligned}$$

Proof. Observe that

$$\Pr\Big[(1-\varepsilon)\sqrt{\frac{n}{t}}D(t) < D(n) < (1+\varepsilon)\sqrt{\frac{n}{t}}D(t) \text{ for all } n \geq t \Bigm| D(t)\Big] \geq 1-(p^+ + p^-)$$

with

$$
\begin{aligned}
p^- &:= \Pr\Big[D(n) < (1-\varepsilon)\sqrt{\frac{n}{t}}D(t) \text{ for some } n \geq t \;\Big|\; D(t)\Big] \\
p^+ &:= \Pr\Big[D(n) > (1+\varepsilon)\sqrt{\frac{n}{t}}D(t) \text{ for some } n \geq t \;\Big|\; D(t)\Big].
\end{aligned}
$$

We proceed by finding upper bounds for p^+ and p^-. For $k \in \mathbf{N}$ let $\delta_k = \frac{\varepsilon}{k^{2/3}}$, $h_k = \prod_{i=1}^{k-1}(1-\delta_i)$, $c_k^- = \prod_{i=1}^{k-1}(1-\frac{1}{2}\delta_i - 2\delta_i^2)$ and $c_k^+ = \prod_{i=1}^{k-1}(1-\frac{1}{2}\delta_i + 2\delta_i^2)$. Every function $f(t) : \mathbf{R} \to \mathbf{R}$ that is a realization of the random variables $D(t)$ is monotonically increasing. It follows using Lemma 14, Lemma 15, the union bound over all possible choices of k, and Lemma 16 that

$$
\begin{aligned}
p^- &\leq \sum_{k=0}^{\infty} \Pr\Big[D((1+\delta_k)h_k t) < (1+\frac{1}{2}\delta_k - 2\delta_k^2)D(h_k t), D(h_k t) \geq c_k^- D(t) \;\Big|\; D(t)\Big] \\
&\qquad \leq \sum_{k=0}^{\infty} e^{-\frac{1}{16}\delta_k^3 c_k^- D(t)}, \\
p^+ &\leq \sum_{k=0}^{\infty} \Pr\Big[D((1+\delta_k)h_k t) > (1+\frac{1}{2}\delta_k + 2\delta_k^2)D(h_k t), D(h_k t) \leq c_k^+ D(t) \;\Big|\; D(t)\Big] \\
&\qquad \leq \sum_{k=0}^{\infty} \log(2et) e^{-\frac{1}{8}\delta_k^3 c_k^+ D(t)}.
\end{aligned}
$$

It remains to show that $p^+ + p^- \leq \log(15t)\varepsilon^{-6}\exp\big(-\varepsilon^{15}10^{-24}D(t)\big)$. This last last step requires a longer calculation which we omit because of space limitations. ◀

The next lemma is a slight variant of Lemma 17. The proof is omitted for lack of space.

▶ **Lemma 18.** *For $t \in \mathbf{R}$, $t \geq 1$, $0 < \varepsilon \leq 1/2$, $d \in \mathbf{N}$ with $\Pr[D(t) = d] \neq 0$ and $d \geq \log(\log(3t))\varepsilon^{-200}$*

$$
\Pr\Big[(1-\varepsilon)\sqrt{\frac{n}{t}}d < D(n) < (1+\varepsilon)\sqrt{\frac{n}{t}}d \textit{ for all } n \geq t \;\Big|\; D(t) = d\Big] \geq 1 - e^{-\varepsilon^{200}d}.
$$

At last, we generalize this result to different values of m.

▶ **Theorem 19.** *For $t, m, d \in \mathbf{N}^+$, $0 < \varepsilon \leq 1/2$, $S \subseteq \{v_1, \dots, v_t\}$ with $\Pr[d_m^t(S) = d] \neq 0$ and $d \geq \log(\log(3tm))\varepsilon^{-200}$*

$$
\Pr\Big[(1-\varepsilon)\sqrt{\frac{n}{t}}d < d_m^n(S) < (1+\varepsilon)\sqrt{\frac{n}{t}}d \textit{ for all } n \geq t \;\Big|\; d_m^t(S) = d\Big] \geq 1 - e^{-\varepsilon^{200}d}.
$$

Proof. As stated in the introduction, we can simulate G_m^n via G_1^{mn}, by merging every m consecutive vertices into a single one. Let G_m^n be a graph with vertices $V = \{v_1, \dots, v_n\}$. We can assume that this graph has been constructed from a graph G_1^{mn} with vertex set $V' = \{v'_1, \dots, v'_{mn}\}$ by merging $v'_{(i-1)m+1}, \dots, v'_{im}$ into v_i for $1 \leq i \leq n$. Let $S' \subseteq V'$ be the set of vertices in G_1^{mn} that are merged into S. Since the graph allows multi-edges, $d_m^n(S)$ and $d_1^{mn}(S')$ have the same probability distribution. Lemma 17 states with $d_1^{mn}(S') = D(mn)$

$$
\Pr\Big[(1-\varepsilon)\sqrt{\frac{n}{t}}d < d_1^{mn}(S') < (1+\varepsilon)\sqrt{\frac{n}{t}}d \text{ for all } nm \geq tm \;\Big|\; d_1^{tm}(S') = d\Big] \geq 1-e^{-\varepsilon^{200}d}. \quad ◀
$$

References

1 Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, 2008.

2 Agnes Backhausz et al. Limit distribution of degrees in random family trees. *Electronic Communications in Probability*, 16:29–37, 2011.

3 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

4 Béla Bollobás and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1):5–34, 2004.

5 Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The Degree Sequence of a Scale-free Random Graph Process. *Random Structures & Algorithms*, 18(3):279–290, May 2001.

6 Anna D Broido and Aaron Clauset. Scale-free networks are rare. *Nature communications*, 10(1):1017, 2019.

7 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David Juedes, Iyad A Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005.

8 Jin Chen, Wynne Hsu, Mong Li Lee, and See-Kiong Ng. NeMoFinder: Dissecting genome-wide protein-protein interactions with meso-scale network motifs. In *Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 106–115. ACM, 2006.

9 Aaron Clauset, Cosma Rohilla Shalizi, and Mark E. J. Newman. Power-Law Distributions in Empirical Data. *SIAM Review*, 51(4):661–703, 2009.

10 Colin Cooper and Alan Frieze. The cover time of the preferential attachment graph. *Journal of Combinatorial Theory, Series B*, 97(2):269–290, 2007.

11 Bruno Courcelle. The Monadic Second-Order Logic of Graphs I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, 1990.

12 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms Are a Good Basis for Counting Small Subgraphs. In *Proc. of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 210–223, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055502.

13 R. Diestel. *Graph Theory*. Springer, Heidelberg, 2010.

14 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

15 Alan Frieze and Colin McDiarmid. Algorithmic theory of random graphs. *Random Structures & Algorithms*, 10(1-2):5–42, 1997.

16 Joshua A Grochow and Manolis Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *Annual International Conference on Research in Computational Molecular Biology*, pages 92–106. Springer, 2007.

17 Svante Janson. Limit theorems for triangular urn schemes. *Probability Theory and Related Fields*, 134(3):417–452, 2006.

18 Svante Janson, Tomasz Łuczak, and Ilkka Norros. Large cliques in a power-law random graph. *Journal of Applied Probability*, 47(4):1124–1135, 2010.

19 Hawoong Jeong, Zoltan Néda, and Albert-László Barabási. Measuring preferential attachment in evolving networks. *EPL (Europhysics Letters)*, 61(4):567, 2003.

20 Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

21 Nitish Korula and Silvio Lattanzi. An efficient reconciliation algorithm for social networks. *Proc. of the VLDB Endowment*, 7(5):377–388, 2014.

22 Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

23 Tamás F Móri. The maximum degree of the Barabási–Albert random tree. *Combinatorics, Probability and Computing*, 14(3):339–348, 2005.

24 Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001.

25 Saeed Omidi, Falk Schreiber, and Ali Masoudi-Nejad. MODA: an efficient algorithm for network motif discovery in biological networks. *Genes & genetic systems*, 84(5):385–395, 2009.

26 Erol Peköz, Adrian Röllin, and Nathan Ross. Joint degree distributions of preferential attachment random graphs. *Advances in Applied Probability*, 49(2):368–387, 2017.

27 Erol A Peköz, Adrian Röllin, Nathan Ross, et al. Degree asymptotics with rates for preferential attachment random graphs. *The Annals of Applied Probability*, 23(3):1188–1218, 2013.

28 Falk Schreiber and Henning Schwöbbermeyer. Frequency concepts and pattern detection for the analysis of motifs in networks. In *Transactions on computational systems biology III*, pages 89–104. Springer, 2005.

29 Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

30 Remco van der Hofstad. *Random graphs and complex networks*, volume 1. Cambridge University Press, 2016.

31 Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(4):347–359, 2006.

32 Panpan Zhang, Chen Chen, and Hosam Mahmoud. Explicit characterization of moments of balanced triangular Pólya urns by an elementary approach. *Statistics & Probability Letters*, 96:149–153, 2015.

Parameterized k-Clustering: Tractability Island

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
Fedor.Fomin@uib.no

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no

Kirill Simonov
Department of Informatics, University of Bergen, Norway
Kirill.Simonov@uib.no

Abstract

In k-Clustering we are given a multiset of n vectors $X \subset \mathbb{Z}^d$ and a nonnegative number D, and we need to decide whether X can be partitioned into k clusters $C_1, \ldots, C_k$ such that the cost

$$\sum_{i=1}^{k} \min_{c_i \in \mathbb{R}^d} \sum_{x \in C_i} \|x - c_i\|_p^p \leq D,$$

where $\| \cdot \|_p$ is the Minkowski (L_p) norm of order p. For $p = 1$, k-Clustering is the well-known k-Median. For $p = 2$, the case of the Euclidean distance, k-Clustering is k-Means. We study k-Clustering from the perspective of parameterized complexity. The problem is known to be NP-hard for $k = 2$ and it is also NP-hard for $d = 2$. It is a long-standing open question, whether the problem is fixed-parameter tractable (FPT) for the combined parameter $d+k$. In this paper, we focus on the parameterization by D. We complement the known negative results by showing that for $p = 0$ and $p = \infty$, k-Clustering is W[1]-hard when parameterized by D. Interestingly, the complexity landscape of the problem appears to be more intricate than expected. We discover a tractability island of k-Clustering: for every $p \in (0, 1]$, k-Clustering is solvable in time $2^{\mathcal{O}(D \log D)}(nd)^{\mathcal{O}(1)}$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases clustering, parameterized complexity, k-means, k-median

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.14

Related Version The full version of this paper is available at `https://arxiv.org/abs/1902.08559`.

Funding This work is supported by the Research Council of Norway via the project "MULTIVAL".

1 Introduction

Recall that for $p > 0$, the *Minkowski* or *L_p-norm* of a vector $x = (x[1], \ldots, x[d]) \in \mathbb{R}^d$ is defined as

$$\|x\|_p = \Big(\sum_{i=1}^{d} |x[i]|^p \Big)^{1/p}.$$

Respectively, we define the *(L_p-norm) distance* between two vectors $x = (x[1], \ldots, x[d])$ and $y = (y[1], \ldots, y[d])$ as

$$\mathrm{dist}_p(x, y) = \|x - y\|_p^p = \sum_{i=1}^{d} |x[i] - y[i]|^p.$$

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 14; pp. 14:1–14:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We also consider dist_p for $p = 0$ and $p = \infty$. For $p = 0$, dist_p is L_0 (or the Hamming) distance, that is the number of different coordinates in x and y:

$$\text{dist}_0(x, y) = |\{i \in \{1, \ldots, d\} \mid x[i] \neq y[i]\}|.$$

For $p = \infty$, dist_p is L_∞-distance, which is defined as

$$\text{dist}_\infty(x, y) = \max_{i \in \{1,\ldots,d\}} |x[i] - y[i]|.$$

The k-Clustering problem is defined as follows. For a given (multi) dataset of n vectors (points) $X \subset \mathbb{Z}^d$, the task is to find a partition of X into k clusters $C_1, \ldots, C_k$ minimizing the cost

$$\sum_{i=1}^{k} \min_{c_i \in \mathbb{R}^d} \sum_{x \in C_i} \text{dist}_p(x, c_i),$$

intuitively, c_i is a centroid of the cluster C_i.

In particular, for $p = 1$, dist_p is the L_1-distance and the corresponding clustering problem is known as k-Median. (Often in the literature, k-Median is also used for clustering minimizing the sums of the Euclidean distances.) For $p = 2$, dist_p is the L_2 (Euclidean) distance, and then the clustering problem becomes k-Means.

Let us note that optimal clusterings for the same set of vectors can be drastically different for various values of p, as shown in Figure 1. As we show in the paper, the complexity of k-Clustering also strongly depends on the choice of p.

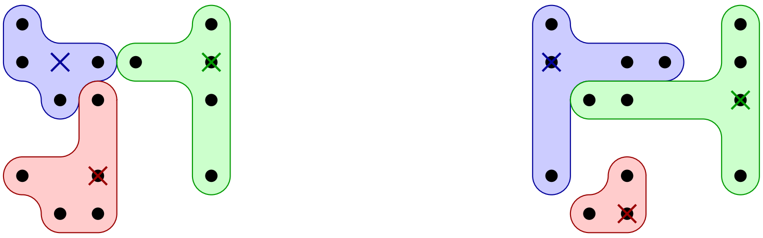


Figure 1 Optimal clusterings of the same set of vectors with different distances: dist_1 in the left subfigure, $\text{dist}_{1/4}$ in the right subfigure. Shapes denote clusters, crosses denote cluster centroids.

k-Clustering, and especially k-Median and k-Means, are among the most prevalent problems occurring in virtually every subarea of data science. We refer to the survey of Jain [22] for an extensive overview. While in practice the most common approaches to clustering are based on different variations of Lloyd's heuristic [25], the problem is interesting from the theoretical perspective as well. In particular, there is a vast amount of literature on approximation algorithms for k-Clustering whose behavior can be analyzed rigorously, see e.g. [1, 2, 6, 8, 9, 16, 17, 20, 24, 13, 23, 10, 30].

When it comes to exact solutions, we observe the following phenomena. While heuristic algorithms for k-Clustering work surprisingly well in practice, from the perspective of the parameterized complexity, k-Clustering is intractable for all previously studied parameterizations, see Table 1. The k-Clustering problem is naturally "multivariate": in addition to the input size n, there are also parameters like space dimension d, number of clusters k or the cost of clustering D. The problem is known to be NP-complete for $k = 2$ [3, 15] and for $d = 2$ [28, 26]. By the classical work of Inaba et al. [21], in the case when both d and k are constants, k-Clustering is solvable in polynomial time $\mathcal{O}(n^{dk+1})$. It is a long-standing open problem whether k-Clustering is FPT parameterized by $d + k$. Under ETH, the lower bound of $n^{\Omega(k)}$, even when $d = 4$, was shown by Cohen-Addad et al. in [11] for the settings where the set of potential candidate centers is explicitly given as input. However the lower bound of Cohen-Addad et al. does not generalize to the settings of this paper when any point in Euclidean space can serve as a center. For the special case, when the input consists of binary vectors and the distance is Hamming, the problem is solvable in time $2^{\mathcal{O}(D \log D)}(nd)^{\mathcal{O}(1)}$ [18].

Our results and approaches. In this paper we investigate the dependence of the complexity of k-Clustering from the cost of clustering D. It appears, that adding this new "dimension" makes the complexity landscape of k-Clustering intricate and interesting. More precisely, we consider the following problem.

k-Clustering with distance dist

Input: A multiset X of n vectors in $\mathbb{Z}^d$, a positive integer k, and a nonnegative number D.

Task: Decide whether there is a partition of X into k clusters $\{C_i\}_{i=1}^k$ and k vectors $\{c_i\}_{i=1}^k$, called *centroids*, in $\mathbb{R}^d$ such that

$$\sum_{i=1}^{k} \sum_{x \in C_i} \operatorname{dist}(x, c_i) \le D.$$

Let us remark that vector set X (like the column set of a matrix) can contain many equal vectors. Also we consider the situation when vectors from X are integer vectors, while centroid vectors are not necessarily from X. Moreover, coordinates of centroids can be reals.

Our main algorithmic result is the following theorem.

▶ **Theorem 1.** *k-Clustering with distance dist_p is solvable in time $2^{\mathcal{O}(D \log D)}(nd)^{\mathcal{O}(1)}$ for every $p \in (0, 1]$.*

Thus k-Clustering when parameterized by D is fixed-parameter tractable (FPT) for Minkowski distance dist_p of order $0 < p \le 1$. In the first step of our algorithm we use color coding to reduce solution of the problem to the Cluster Selection problem, which we find interesting on its own. In Cluster Selection we have t groups of weighted vectors and the task is to select exactly one vector from each group such that the weighted cost of the composite cluster is at most D. More formally,

Cluster Selection with distance dist

Input: A set of m vectors X given together with a partition $X = X_1 \cup \cdots \cup X_t$ into t disjoint sets, a weight function $w : X \to \mathbb{Z}_+$, and a nonnegative number D.

Task: Decide whether it is possible to select exactly one vector x_i from each set X_i such that the total cost of the composite cluster formed by $x_1, \ldots, x_t$ is at most D:

$$\min_{c \in \mathbb{R}^d} \sum_{i=1}^{t} w(x_i) \cdot \operatorname{dist}(x_i, c) \le D.$$

The Cluster Selection problem is closely related to variants of the well-known Consensus Pattern problem. Namely, for the Hamming distance, the definition of Cluster Selection nearly coincides with the Colored Consensus Strings with Outliers problem studied in [7], only in the latter the alphabet is assumed to be of constant size.

Informally (see Theorem 10 for the precise statement), our reduction shows that if the distance norm satisfies some specific properties (which dist_p satisfies for all p) and if Cluster Selection is FPT parameterized by D, then so is k-Clustering. Therefore, in order to prove Theorem 1, all we need is to show that Cluster Selection is FPT parameterized by D when $p \in (0, 1]$. This is the most difficult part of the proof. Here we invoke the theorem of Marx [27] on the number of subhypergraphs in hypergraphs of bounded fractional edge cover.

Superficially, the general idea of the proof of Theorem 1 is similar to the idea behind the algorithm for Binary r-Means for L_0 from [18]. In both cases, the classical color coding technique of Alon et al. [4] is used as a preprocessing step. However, the further steps in [18] strongly exploit the fact that the data is binary. As we will see in Theorem 2, the existence of an FPT algorithm for k-Clustering in L_0 is highly unlikely. Thus the reductions from [18] cannot be applied in our case, and we need a new approach.

More precisely, for clustering in L_0 we prove the following theorem.

▶ **Theorem 2.** *With distance* dist_0*, k-*Clustering *parameterized by $d + D$ and* Cluster Selection *parameterized by $d + t + D$ are W[1]-hard.*

In particular, this means that up to a widely-believed assumption in complexity that FPT ≠ W[1], Theorem 2 rules out algorithms solving k-Clustering in time $f(d, D) \cdot n^{\mathcal{O}(1)}$ and algorithms solving Cluster Selection in L_0 in time $g(t, d, D) \cdot n^{\mathcal{O}(1)}$ for any functions $f(d, D)$ and $g(t, d, D)$. A similar hardness result holds for L_∞.

▶ **Theorem 3.** *With distance* dist_∞*, k-*Clustering *parameterized by D and* Cluster Selection *parameterized by $t + D$ are W[1]-hard.*

This naturally brings us to the question: What happens with k-Clustering for $p \in (1, \infty)$, especially for the Euclidean distance, that is $p = 2$. Unfortunately, we are not able to answer this question when the parameter is D only. However, we can prove that

▶ **Theorem 4.** *k-*Clustering *and* Cluster Selection *with distance* dist_2 *are FPT when parameterized by $d + D$.*

Thus in particular, Theorem 4 implies that k-Clustering with distance dist_2 is FPT parameterized by $d + D$. On the other hand, we prove that

▶ **Theorem 5.** Cluster Selection *with distance* dist_p *is W[1]-hard for every $p \in (1, \infty)$ when parameterized by $t + D$.*

In particular, Theorem 5 yields that the approach we used to establish the tractability (with parameter D) of k-Clustering for $p = 1$ will not work for $p > 1$.

We summarize our and previously known algorithmic and hardness results for the problems k-Clustering and Cluster Selection with different distances in Table 1.

In the extended abstract, we provide a full proof of Theorems 1 and 15. Proofs of Theorems 2, 3, 4, 5, 19 and 28 can be found in the full version of this paper [19].

The remaining part of this paper is organized as follows. Section 2 contains preliminaries. In Section 3 we prove Theorem 10 which provides us with FPT Turing reduction from k-Clustering to Cluster Selection. Theorem 10 appears to be a handy tool to establish

Table 1 Complexity of k-Clustering and Cluster Selection.

dist_p	k-Clustering	Cluster Selection
$p = 0$	W[1]-hard param. $d + D$ [Thm 2] NP-c for $k = 2$ [15]	W[1]-hard param. $d + t + D$ [Thm 2]
$0 < p \leq 1$	$2^{\mathcal{O}(D \log D)}(nd)^{\mathcal{O}(1)}$ [Thm 1] NP-c for $k = 2$ when $p = 1$ [15] NP-c for $d = 2$ when $p = 1$ [28]	$2^{\mathcal{O}(D \log D)}(nd)^{\mathcal{O}(1)}$ [Thm 15] W[1]-hard param. $t + d$ for $p = 1$ [Thm 19]
$1 < p < \infty$	FPT param. $d + D$ for $p = 2$ [Thm 4] NP-c for $k = 2$ when $p = 2$ [3] NP-c for $d = 2$ when $p = 2$ [26]	FPT param. $d + D$ for $p = 2$ [Thm 4] W[1]-hard param. $t + D$ [Thm 5]
$p = \infty$	W[1]-hard param. D [Thm 3] NP-c for $k = 2$ [Thm 28]	W[1]-hard param. $t + D$ [Thm 3]

tractability of k-Clustering. In Section 4 we prove Theorem 1, the main algorithmic result of this work, stating that when $p \in (0, 1]$, k-Clustering and Cluster Selection admit FPT algorithms with parameter D. We conclude with open problems in Section 5.

2 Preliminaries and notation

Cluster notation. By a *cluster* we always mean a multiset of vectors in $\mathbb{Z}^d$. For distance dist, the *cost* of a given cluster C is the total distance from all vectors in the cluster to the optimally selected cluster centroid, $\min_{c \in \mathbb{R}^d} \sum_{x \in C} \text{dist}(x, c)$. An *optimal* cluster centroid for a given cluster C is any $c \in \mathbb{R}^d$ minimizing $\sum_{x \in C} \text{dist}(x, c)$. For most of the considered distances, we argue that an optimal cluster centroid could always be chosen among selected family of vectors (e.g. integral). Whenever we show this, we only consider optimal cluster centroids of the stated form afterwards.

Complexity. A *parameterized problem* is a language $Q \subseteq \Sigma^* \times \mathbb{N}$ where Σ^* is the set of strings over a finite alphabet Σ. Respectively, an input of Q is a pair (I, k) where $I \subseteq \Sigma^*$ and $k \in \mathbb{N}$; k is the *parameter* of the problem. A parameterized problem Q is *fixed-parameter tractable* (FPT) if it can be decided whether $(I, k) \in Q$ in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for some function f that depends of the parameter k only. Respectively, the parameterized complexity class FPT is composed by fixed-parameter tractable problems. The W-hierarchy is a collection of computational complexity classes: we omit the technical definitions here. The following relation is known amongst the classes in the W-hierarchy: $\mathsf{FPT} = \mathsf{W}[0] \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \ldots \subseteq \mathsf{W}[P]$. It is widely believed that $\mathsf{FPT} \neq \mathsf{W}[1]$, and hence if a problem is hard for the class $\mathsf{W}[i]$ (for any $i \geq 1$) then it is considered to be fixed-parameter intractable. We refer to books [12, 14] for the detailed introduction to parameterized complexity.

Real computations. As is usual in computational geometry, we express the running time of algorithms in terms of number of operations over the reals. This is natural since to compute L_p-distances we have to deal with numbers of form x^p where x is an integer and p is any real number. However, in special cases the bounds hold even for more restrictive models, e.g. when $p = 1$ or $p = 2$ the algorithms operate only on integers of polynomially bounded length.

3 From k-Clustering to Cluster Selection

In this section we present a general scheme for obtaining an FPT algorithm parameterized by D, which is later applied to various distances.

First, we formalize the following intuition: there is no reason to assign equal vectors to different clusters.

▶ **Definition 6** (Initial cluster and regular partition). *For a multiset of vectors X, an inclusion-wise maximal multiset $I \subset X$ such that all vectors in I are equal is called* an initial cluster*.*

We say that a clustering $\{C_1, \ldots, C_k\}$ of X is regular *if for every initial cluster I there is a $i \in \{1, \ldots, k\}$ such that $I \subset C_i$.*

Now we prove that it suffices to look only for regular solutions.

▶ **Proposition 7.** *Let (X, k, D) be a yes-instance to k-*Clustering*. Then there exists a solution of (X, k, D) which is a regular clustering.*

Proof. Let us assume that the instance (X, k, D) has a solution. There are k clusters $\{C_i\}_{i=1}^k$ and k vectors $\{c_i\}_{i=1}^k$ in $\mathbb{R}^d$ such that $\sum_{i=1}^k \sum_{x \in C_i} \operatorname{dist}(x, c_i) \le D$. Note that for every $x \in C_j$, $\operatorname{dist}(x, c_j) \ge \min_{1 \le i \le k} \operatorname{dist}(x, c_i)$. So if we consider a new clustering $\{C'_1, \ldots, C'_k\}$ with the same centroids, where C'_j are all vectors from X for which c_j is the closest centroid, the total distance does not increase. If we also break ties in favor of the lower index, then for any initial cluster I the same centroid c_i will be the closest, and all vectors from I will end up in C'_i, so $\{C'_1, \ldots, C'_k\}$ is a regular clustering. ◀

From now on, we consider only regular solutions.

▶ **Definition 8** (Simple and composite clusters). *We say that a cluster C is* simple *if it is an initial cluster. Otherwise, the cluster is* composite*.*

Next we state a property of k-Clustering with a particular distance, which is required for the algorithm. Intuitively, each unique vector adds at least some constant to the cluster cost. In the subsequent sections we show that the property holds for all distances in our consideration.

▶ **Definition 9** (α-property). *We say that a distance has the α-*property *for some $\alpha > 0$ if for any s the cost of any composite cluster which consists of s initial clusters is at least $\alpha(s-1)$.*

The Cluster Selection problem defined in the introduction is a key subroutine in our algorithm. In some cases the problem is solvable trivially, but it presents the main challenge for our main algorithmic result with the L_1 distance. The intuition to the weight function in the definition of Cluster Selection is that it represents sizes of initial clusters, that is, how many equal vectors are there.

We also need a procedure to enumerate all possible optimal cluster costs which are less than D. It may not be straightforward since not all distances in our consideration are integer. So we assume that the set of all possible optimal cluster costs which are less than D is also given in the input. Now we are ready to state the result formally.

▶ **Theorem 10.** *Assume that the α-property holds,* Cluster Selection *is solvable in time $\Phi(m, d, t, D)$, where Φ is a non-decreasing function of its arguments, and we are given the set $\mathcal{D}$ of all possible optimal cluster costs which are at most D. Then k-*Clustering *is solvable in time*

$$2^{\mathcal{O}(D \log D)} (nd)^{\mathcal{O}(1)} |\mathcal{D}| \Phi(n, d, 2D/\alpha, D).$$

Proof. By the α-property, in any solution there are at most D/α composite clusters, since each contains at least two initial clusters. Moreover, there are at most $2D/\alpha$ initial clusters in all composite clusters.

Thus by Proposition 7, solving k-Clustering is equivalent to selecting at most $T := \lceil 2D/\alpha \rceil$ initial clusters and grouping them into composite clusters such that the total cost of these clusters is at most D. We design an algorithm which, taking as a subroutine an algorithm for Cluster Selection, solves k-Clustering. An example is shown in Figure 2.

To perform the selection and grouping, our algorithm uses the color coding technique of Alon, Yuster, and Zwick from [4]. Consider the input as a family of initial clusters $\mathcal{I}$. We color initial clusters from $\mathcal{I}$ independently and uniformly at random by T colors 1, 2, …, T. Consider any solution, and the particular set of at most T initial clusters which are included into composite clusters in this solution. These initial clusters are colored by distinct colors with probability at least $\frac{T!}{T^T} \geq e^{-T}$. Now we construct an algorithm for finding a colorful solution.

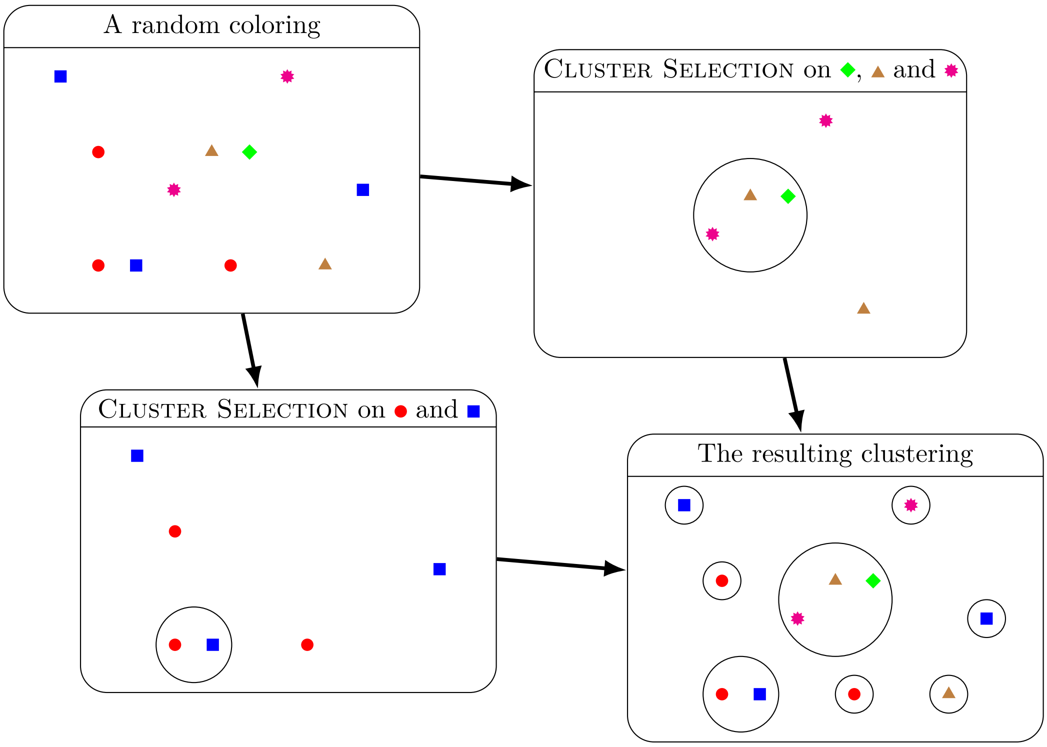


Figure 2 An illustration to the algorithm in Theorem 10. We start with a particular random coloring and a particular partition of colors $\mathcal{P} = \{P_1, P_2\}$, where $P_1 = \{$●,■$\}$ and $P_2 = \{$◆,▲,✸$\}$. We make two calls to Cluster Selection with respect to P_1 and P_2 and construct the resulting clustering. In the example, all input vectors are distinct.

We consider all possible ways to split colors between clusters (some colors may be unused). Hence we consider all possible families $\mathcal{P} = \{P_1, \ldots, P_h\}$ of pairwise disjoint non-empty subsets of $\{c \in \{1, \ldots, T\} :$ there exists $J \in \mathcal{I}$ colored by $c\}$. Each family $\mathcal{P}$ corresponds to a partition of the set of colors $\{1, \ldots, T\}$ if we add one fictitious subset for colors which are not used in the composite clusters. The total number of partitions does not exceed $T^T = 2^{\mathcal{O}(D \log D)}$.

When partition $\mathcal{P}$ is fixed, we form clusters by solving instances of Cluster Selection: For each $i \in \{1, \dots, h\}$, we take initial clusters colored by elements of P_i, bundle together those with the same color, and pass the resulting family to Cluster Selection. First note that there cannot be $P \in \mathcal{P}$ of size at most one, since then Cluster Selection has to make a simple cluster while we assume that all clusters obtained from $\mathcal{P}$ are composite. Second, the total number of clusters has to be k, the number of clusters is $|\mathcal{I}| - \sum_{P \in \mathcal{P}} |P| + |\mathcal{P}|$. For each $\mathcal{P}$ we check that both conditions hold, and if not, we discard the choice of $\mathcal{P}$ and move to the next one, before calling the Cluster Selection subroutine.

Next, we formalize how we call the Cluster Selection subroutine. We fix the set of colors $P_i = \{c_1, \dots, c_t\}$, then take the sets $I_j = \{J \in \mathcal{I} : J \text{ is colored by } c_j\}$ for $j \in \{1, \dots, t\}$. We turn each set of initial clusters I_j into a set of weighted vectors X_j naturally: For each $J \in I_j$, we put one vector $x \in J$ into X_j, and $w(x) := |J|$. The family of sets of vectors X_1, $\dots$, X_t and the weight function w are the input for Cluster Selection. Then we search for the minimum cluster cost bound $d_i \leq D$ from $\mathcal{D}$, for which the instance $(X_1, \dots, X_t, d_i)$ of Cluster Selection is a yes-instance, running each time the algorithm for Cluster Selection.

If for some i setting d_i to D leads to a no-instance, or if $\sum_{i=1}^{h} d_i > D$, then we discard the choice of the partition $\mathcal{P}$ and move to the next one. Otherwise, we report that k-Clustering has a solution and stop. Next, we prove that in this case the solution indeed exists.

We reconstruct the solution to k-Clustering as follows: For each $i \in \{1, \dots, h\}$ the corresponding to $P_i = \{c_1, \dots, c_t\}$ instance of Cluster Selection has a solution $\{x_1, \dots, x_t\}$. For each $j \in \{1, \dots, t\}$, consider the corresponding initial cluster J_j consisting of $w(x_j)$ vectors equal to x_j. For each $i \in \{1, \dots, h\}$ we obtain a composite cluster $\cup_{j=1}^{t} J_j$, all other clusters are simple. So the total cost is $\sum_{i=1}^{h} d_i$, which is at most D. Thus, if the algorithm finds a solution, then (X, d, D) is a yes-instance.

In the opposite direction. If there is a solution to k-Clustering, then there is a regular solution, and with probability at least e^{-T} initial clusters which are parts of composite clusters in this solution are colored by distinct colors. Then, there is a partition $\mathcal{P} = \{P_1, \dots, P_h\}$ which corresponds to this solution. This partition is obtained as follows: put into P_1 colors from the first composite cluster, into P_2 from the second and so on. At some point our algorithm checks the partition $\mathcal{P}$, and as it finds the optimal cost value for each cluster, then it is at most the cost of the corresponding cluster of the solution from which we started.

To analyze the running time, we consider $2^{\mathcal{O}(D \log D)}$ partitions $\mathcal{P}$, for each $\mathcal{P}$ we $|\mathcal{P}| = \mathcal{O}(D)$ times search for optimal d_i. And for each of $|\mathcal{D}|$ possible values [1] of d_i we make one call to the Cluster Selection algorithm, which takes time at most $\Phi(n, d, T, D)$.

To amplify the error probability to be at least $1/e$, we do $N = \lceil e^T \rceil$ iterations of the algorithm, each time with a new random coloring. As each iteration succeeds with probability at least e^{-T}, the probability of not finding a colorful solution after N iterations is at most $(1 - e^{-T})^{e^T} \leq e^{-1} < 1$. So the total running time is $2^{\mathcal{O}(D \log D)} \cdot (nd)^{\mathcal{O}(1)} |\mathcal{D}| \Phi(n, d, 2D/\alpha, D)$.

The algorithm could be derandomized by the standard derandomization technique using perfect hash families [4, 29]. So k-Clustering is solvable in the same deterministic time. ◀

[1] We could also binary search for the optimal $d_i \in \mathcal{D}$ instead, thus replacing $|\mathcal{D}|$ by $\log |\mathcal{D}|$ in the running time. However, for all choices of $\mathcal{D}$ we consider this does not make a difference.

4 Algorithms and complexity for distances with $p \in (0, 1]$

The main motivation for the results in this section is the study of k-Clustering with the L_1 distance, the case widely known as k-Medians. However, our main algorithmic result also extends to distances of order $p \in (0, 1)$ since in some sense they behave similarly to the L_1 distance.

4.1 FPT algorithm when parameterized by D

In this subsection, we prove Theorem 1: when $p \in (0, 1]$, k-Clustering admits an FPT algorithm with parameter D. First we state basic geometrical observations for cases $p = 1$ and $p \in (0, 1)$, Then we propose a general algorithm for Cluster Selection which relies only on these properties. Finally, we show how Theorem 10 could be applied.

The next two claims deal with the structure of optimal cluster centroids. We state and prove them in the case of weighted vectors where each vector has a positive integer weight given by a weight function w. The unweighted case is just a special case when the weight of each vector is one. The proofs of the claims are straightforward and are available in the full version of this paper.

First, we show that coordinates of cluster centroids could always be selected among the values present in the input, which helps greatly in enumerating cluster centroids that may be optimal.

▷ Claim 11. Assume $p \in (0, 1]$, let $C = \{x_1, \ldots, x_t\}$ be a cluster and $w : \{x_1, \cdots, x_t\} \to \mathbb{Z}_+$ be a weight function. There is an optimal (subject to the weighted distance $w(x_i) \cdot \text{dist}_p(x_i, c)$) centroid c of C such that for each $i \in \{1, \ldots, d\}$, the i-th coordinate $c[i]$ of the centroid is from the values present in the input in this coordinate, that is $c[i] \in \{x_1[i], \ldots, x_t[i]\}$. Moreover, for $p = 1$ we may assume that the optimal value is a weighted median of the values present in the i-th coordinate.

In particular, by Claim 11 we may assume that the coordinates of optimal cluster centroids are integers. Then, the α-property holds with $\alpha = 1$ since at most one of the initial clusters could have distance zero to the cluster centroid, and all others have distance at least one since the cluster centroid is integral. Namely, let x be a vector in the cluster, and c be the cluster centroid, if $x \neq c$, then there is a coordinate j where x and c differ, and since they are both integral, $|x[j] - c[j]| \geq 1$, and

$$\text{dist}_p(x, c) = \sum_{i=1}^{d} |x[i] - c[i]|^p \geq |x[j] - c[j]|^p \geq 1^p = 1.$$

In what follows, the expression *half of vectors by weight* means that the total weight of the corresponding set of vectors is at least half of the total weight of C.

▷ Claim 12. If at least half of the vectors by weight in the cluster C have the same value z in some coordinate i, then the optimal cluster centroid is also equal to z in this coordinate.

In order to apply Theorem 10, we need an FPT algorithm for Cluster Selection. Before obtaining it, we state some properties of hypergraphs, which we need for the algorithm.

A *hypergraph* G is a set of vertices $V(G)$ and a collection of hyperedges $E(G)$, each hyperedge is a subset of $V(G)$. If G and H are hypergraphs, we say that H *appears* at $V' \subset V(G)$ as a *subhypergraph* if there is a bijection $\pi : V(H) \to V'$ with a property that for any $E \in E(H)$ there is $E' \in E(G)$ such that $\pi(E) = E' \cap V'$, the action of π is extended to subsets of $V(H)$ in a natural way.

A *fractional edge cover* of a hypergraph H is an assignment $\psi : E(H) \to [0,1]$ such that for every $v \in V(H)$, $\sum_{E\in E(H):v\in E} \psi(E) \geq 1$. The *fractional cover number* $\rho^*(H)$ is the minimum of $\sum_{E\in E(H)} \psi(E)$ taken over all fractional edge covers ψ.

We need the following result of Marx [27] about finding occurences of one hypergraph in another.

▶ **Lemma 13** ([27]). *Let H be a hypergraph with fractional cover number $\rho^*(H)$, and let G be a hypergraph where each hyperedge has size at most ℓ. There is an algorithm that enumerates in time $|V(H)|^{\mathcal{O}(|V(H)|)} \cdot \ell^{|V(H)|\rho^*(H)+1} \cdot |E(G)|^{\rho^*(H)+1} \cdot |V(G)|^2$ every subset $V' \subset V(G)$ where H appears in G as a subhypergraph.*

Also, the following version of the Chernoff Bound will be of use.

▶ **Proposition 14** ([5]). *Let X_1, X_2, ..., X_n be independent 0-1 random variables. Denote $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Then for $0 < \beta \leq 1$,*

$$P[X \leq (1-\beta)\mu] \leq \exp(-\beta^2\mu/2),$$
$$P[X \geq (1+\beta)\mu] \leq \exp(-\beta^2\mu/3).$$

We are ready to proceed with the proof that Cluster Selection with $p \in (0,1]$ is FPT when parameterized by D.

▶ **Theorem 15.** *For every $p \in (0,1]$,* Cluster Selection *with distance* dist_p *is solvable in time $2^{\mathcal{O}(D\log D)}(md)^{\mathcal{O}(1)}$.*

Proof. First we check if any of the given vectors could be the centroid of the resulting composite cluster. When the centroid is fixed, we find the optimal solution in polynomial time by just selecting the cheapest vector with respect to this centroid from each set. If at some point we find a suitable centroid, then we return that the solution exists. If not, we may assume that the centroid is not equal to any of the given vectors. As a consequence, any vector x selected into the solution cluster contributes at least $w(x)$ to the total distance, since the centroid must be integral by Claim 11. So we may now consider only vectors of weight at most D and, moreover, the total weight of the resulting cluster is at most D.

Consider a resulting cluster C with the centroid c. There is some x_1 in C from X_1, and $\mathrm{dist}_p(x_1, c) \leq D$. So if we try all possible x_1 from X_1 (there are at most m of them), any feasible centroid is at distance at most D from at least one of them. Since x_1 and c are integral, they could be different in at most D coordinates, as $\mathrm{dist}_p(x_1, c) = \sum_{i=1}^{d} |x_1[i] - c[i]|^p \leq D$.

We try all possible $x_1 \in X_1$. After x_1 is fixed, we enumerate all subsets P of coordinates $\{1, \ldots, d\}$ where x_1 and c could differ, we show how to do it efficiently afterwards. When the subset of coordinates P is fixed, we consider all possible centroids, which are integral, equal to x_1 in all coordinates except P, and differ from x_1 by at most $D^{1/p}$ in each of coordinates from P. If $|x_1[i^*] - c[i^*]| > D^{1/p}$ for some coordinate i^*, then $\mathrm{dist}_p(x_1, c) = \sum_{i=1}^{d} |x_1[i] - c[i]|^p \geq |x_1[i^*] - c[i^*]|^p > D$, so c can not be a centroid. With restrictions stated above, there are at most $2^{\mathcal{O}(D\log D)}$ possible centroids.

It remains to show that we could enumerate all possible coordinate subsets efficiently. We reduce this task to the task of finding a specific subhypergraph and then apply Lemma 13.

▷ Claim 16. There are $2^{\mathcal{O}(D\log D)}$ coordinate subsets where x_1 and an optimal cluster centroid c could differ. There exists an algorithm which enumerates all of them in time $2^{\mathcal{O}(D\log D)}(md)^{\mathcal{O}(1)}$.

Proof. Let G be a hypergraph with $V(G) = \{1, \ldots, d\}$, one vertex for each coordinate, and for each vector x in $\cup_{j=1}^{t} X_j$ we take $w(x)$ multiple hyperedges E_x which contains exactly the coordinates where x and x_1 differ. We add an edge only if there are at most D such coordinates, otherwise x can not be in the same cluster as x_1. So hyperdeges in G are of size at most D. Since we consider only vectors of weight at most D, $|E(G)| \leq Dm$.

For a solution, let x_j be the vector selected from the corresponding X_j, for $j \in \{1, \ldots, t\}$, $C = \{x_1, \ldots, x_t\}$ be the solution cluster and c be the centroid. All vectors in C are identical in all coordinates except at most D, since if there are different values in at least $D + 1$ coordinates, the cost is at least $D + 1$. Denote this subset of coordinates as Q, c could also differ from x_1 only at Q. Denote the subset of coordinates where c differs from x_1 as P, $P \subset Q$ and so $|P| \leq D$. The solution (C, c) induces a subhypergraph H of G in the following way. Leave only hyperedges corresponding to the vectors in C, and restrict them to vertices in P. There are at most D vertices and at most D hyperedges in H, since the total weight is at most D. An example of the correspondence between input vectors and hypergraphs is given in Figure 3.

$D = 2$

v	1	2	3	4	5
x_1	0	2	1	3	2
x_2	0	**1**	1	3	**1**
x_3	**1**	2	1	3	**1**
x_4	0	2	**2**	3	2
x_5	0	2	**2**	3	**1**
c	0	2	**2**	3	2

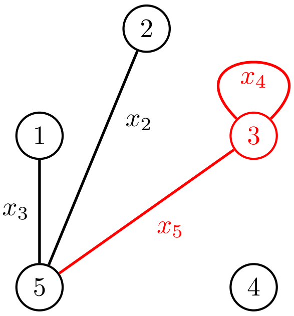


Figure 3 An illustration of the hypergraph construction in Claim 16. On the left, the vector x_1 and other input vectors $x_2, \ldots, x_5$ are given. On the right, the corresponding hypergraph G. The solution is in red: on the left, the resulting cluster $\{x_1, x_4, x_5\}$ is of cost 2; on the right, the corresponding subhypergraph is H. Note that in H the hyperedge x_5 is restricted to the only vertex 3, so its size is one.

The next claim shows that the fractional cover number of H is bounded by a constant.

▷ Claim 17. Each vertex in H is covered by at least half of the hyperedges of H, and $\rho^*(H) \leq 2$.

Proof. Consider a vertex $p \in P$, and assume that less than half of the hyperedges cover p. It means that in the p-th coordinate the centroid c differs from x_1, but less than half of the vectors in C by weight differ from x_1 in this coordinate. This contradicts Claim 12.

So each vertex is covered by at least half of the hyperedges, and setting $\psi \equiv \frac{2}{|E(H)|}$ leads to $\rho^*(H) \leq 2$. ◁

In order to enumerate all possible subsets of coordinates P, we try all hypergraphs H with at most D vertices and at most D hyperedges, and if each vertex is covered by at least half of the hyperedges, we find all places where H appears in G by Lemma 13. The last step is done in $2^{\mathcal{O}(D \log D)} \cdot (md)^{\mathcal{O}(1)}$ time. However, the number of possible H could be up to $2^{\Omega(D^2)}$. The following claim, which is analogous to Proposition 6.3 in [27], shows that we could consider only hypergraphs with a logarithmic number of hyperedges.

▷ Claim 18. If $D \geq 2$, it is possible to delete all except at most $160 \ln D$ hyperedges from H so that in the resulting hypergraph H^* each vertex is covered by at least $1/4$ of the hyperedges, and $\rho^*(H^*) \leq 4$.

Proof. Denote $s = |E(H)|$, construct a new hypergraph H^* on the same vertex set $V(H)$ by independently selecting each hyperedge of H with probability $(120 \ln D)/s$. Applying Proposition 14 with $\beta = 1/3$, probability of selecting more than $160 \ln D$ hyperedges is at most $\exp((-120 \ln D)/27) < 1/D^2$. By Claim 17, each vertex v of H is covered by at least $s/2$ hyperedges, and the expected number of hyperedges covering v in H^* is at least $60 \ln D$. By Proposition 14 with $\beta = 1/3$, the probability that v is covered by less than $40 \ln D$ hyperedges in H^* is at most $\exp(-60 \ln D/18) \leq 1/D^3$. By the union bound, with probability at least $1 - 1/D^2 - D \cdot 1/D^3 > 0$ we select at most $160 \ln D$ hyperedges and each vertex is covered by at least $40 \ln D$ hyperedges. So the claim holds, and $\rho^*(H^*) \leq 4$ by setting $\psi \equiv \frac{4}{|E(H^*)|}$. ◁

Thus, if there is a subhypergraph H in G corresponding to a solution, then there is also a subhypergraph H^* in G appearing at the same subset of $V(G)$ with at most $160 \ln D$ hyperedges and where each vertex is covered by at least $1/4$ of the hyperedges. Since we only need to enumerate possible coordinate subsets, in our algorithm we try all hypergraphs of this form and apply Lemma 13 for each of them. Since there are at most $2^{\mathcal{O}(D \log D)}$ hypergraphs with at most $160 \ln D$ hyperedges and since the fractional cover number is still bounded by a constant, the total running time is $2^{\mathcal{O}(D \log D)} \cdot (md)^{\mathcal{O}(1)}$, as desired. ◁

With Claim 16 proven, the proof of the theorem is complete. ◀

Combining Theorem 10 and Theorem 15, we obtain an FPT algorithm for k-Clustering. This proves Theorem 1, which we recall here.

▶ **Theorem 1.** *k-Clustering with distance dist_p is solvable in time $2^{\mathcal{O}(D \log D)}(nd)^{\mathcal{O}(1)}$ for every $p \in (0, 1]$.*

Proof. We have an algorithm for Cluster Selection whose running time is specified by Theorem 15. By Claim 11, the α-property holds. The only missing part is to describe the way of producing the set $\mathcal{D}$ of all possible cluster costs which are at most D.

In the case $p = 1$ all distances are integral so we can take $\mathcal{D} = \{0, \dots, D\}$.

For the general case, let $\mathcal{B} = \{a^p : a \in \{1, \dots, \lceil D^{1/p} \rceil\}\}$. Consider a cluster $C = \{x_1, \dots, x_t\}$ and the corresponding optimal cluster centroid c. For any $x_j \in C$, $\mathrm{dist}_p(x_j, c) = \sum_{i=1}^d |x_j[i] - c[i]|^p$ is a combination of elements of $\mathcal{B}$ with nonnegative integer coefficients. This is because x_j and c are integral and the cluster cost is at most D, hence $|x_j[i] - c[i]| \leq D^{1/p}$ for each $i \in \{1, \dots, d\}$. Since weights are also integral, the whole cluster cost is a combination of distances between cluster vectors and the centroid with nonnegative integer coefficients, and so also a combination of elements of $\mathcal{B}$ with nonnegative integer coefficients. This means that we can take

$$\mathcal{D} = \left\{ \sum_{b \in \mathcal{B}} a_b \cdot b : a_b \in \mathbb{Z}, a_b \geq 0, \sum_{b \in \mathcal{B}} a_b \leq D \right\},$$

the sum of coefficients a_b is at most D since all elements of $\mathcal{B}$ are at least 1. The size of $\mathcal{D}$ is at most $|\mathcal{B}|^D = 2^{\mathcal{O}(D \log D)}$. ◀

Note that another widely studied version of k-Clustering is where centroids c_i could be selected only among the set of given vectors. Naturally, our algorithm also works in this setting since the set of possible centroids is only restricted further.

Also note that Claim 11 and Claim 12 do not hold in the case $p > 1$, and our algorithm relies heavily on the structure provided by them. Therefore, it does not seem that the algorithm could be extended to the case $p > 1$.

5 Conclusion and open problems

In this paper, we presented an FPT algorithm for k-Clustering with $p \in (0, 1]$ parameterized by D. However, for the case $p \in (1, \infty)$ we were able only to show the W[1]-hardness of Cluster Selection. While intractability of Cluster Selection does not exclude that k-Clustering could be FPT with $p \in (1, \infty)$, it indicates that the proof of this (if it is true at all) would require an approach completely different from ours. Thus an interesting and very concrete open question concerns the parameterized complexity of k-Clustering with $p \in (1, \infty)$ and parameter D.

Another open question is about the fine-grained complexity of k-Clustering when parameterized by $k + d$. For several distances, we know XP-algorithms: an $\mathcal{O}(n^{dk+1})$ algorithm by Inaba et. al. [21] for $p = 2$, as well as trivial algorithms for $p \in [0, 1]$. For the case when the possible cluster centroids are given in the input, the matching lower bound is shown in [11]. However, we are not aware of a lower bound complementing the algorithmic results in the case when any point in Euclidean space can serve as a centroid.

References

1 Marcel R. Ackermann, Johannes Blömer, and Christian Sohler. Clustering for metric and nonmetric distance measures. *ACM Trans. Algorithms*, 6(4):59:1–59:26, 2010. doi:10.1145/1824777.1824779.

2 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004. doi:10.1145/1008731.1008736.

3 Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, May 2009. doi:10.1007/s10994-009-5103-0.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

5 D. Angluin and L.G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *J. Computer and System Sciences*, 18(2):155–193, 1979. doi:10.1016/0022-0000(79)90045-X.

6 Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 250–257. ACM, 2002. doi:10.1145/509907.509947.

7 Christina Boucher, Christine Lo, and Daniel Lokshantov. Consensus Patterns (Probably) Has no EPTAS. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015*, pages 239–250, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

8 Christos Boutsidis, Anastasios Zouzias, Michael W. Mahoney, and Petros Drineas. Randomized Dimensionality Reduction for k-Means Clustering. *IEEE Trans. Information Theory*, 61(2):1045–1062, 2015. doi:10.1109/TIT.2014.2375327.

9 Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the 47tg annual ACM symposium on Theory of Computing (STOC)*, pages 163–172. ACM, 2015.

10 Vincent Cohen-Addad. A Fast Approximation Scheme for Low-dimensional K-means. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 430–440. SIAM, 2018. URL: http://dl.acm.org/citation.cfm?id=3174304.3175298.

11 Vincent Cohen-Addad, Arnaud de Mesmay, Eva Rotenberg, and Alan Roytman. The Bane of Low-dimensionality Clustering. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 441–456. SIAM, 2018. URL: http://dl.acm.org/citation.cfm?id=3174304.3175300.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 W. Fernandez de la Vega, Marek Karpinski, Claire Kenyon, and Yuval Rabani. Approximation Schemes for Clustering Problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 50–58. ACM, 2003. doi:10.1145/780542.780550.

14 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

15 Uriel Feige. NP-hardness of hypercube 2-segmentation. *CoRR*, abs/1411.0821, 2014. arXiv:1411.0821.

16 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 569–578. ACM, 2011. doi:10.1145/1993636.1993712.

17 Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1434–1453. SIAM, 2013. doi:10.1137/1.9781611973105.

18 Fedor V. Fomin, Petr A. Golovach, and Fahad Panolan. Parameterized Low-Rank Binary Matrix Approximation. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.53.

19 Fedor V. Fomin, Petr A. Golovach, and Kirill Simonov. Parameterized k-Clustering: The distance matters!, 2019. arXiv:1902.08559.

20 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 291–300. ACM, 2004.

21 Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the 10th annual Symposium on Computational Geometry (SoCG)*, pages 332–339. ACM, 1994.

22 Anil K Jain. Data clustering: 50 years beyond K-means. *Pattern recognition letters*, 31(8):651–666, 2010.

23 Stavros G. Kolliopoulos and Satish Rao. A Nearly Linear-Time Approximation Scheme for the Euclidean k-Median Problem. *SIAM J. Computing*, 37(3):757–782, June 2007. doi:10.1137/S0097539702404055.

24 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, 2010. doi:10.1145/1667053.1667054.

25 Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Information Theory*, 28(2):129–136, 1982. doi:10.1109/TIT.1982.1056489.

26 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The Planar k-Means Problem is NP-Hard. In *Proceedings of the 3rd International Workshop on Algorithms and Computation (WALCOM)*, Lecture Notes in Comput. Sci., pages 274–285. Springer, 2009. doi:10.1007/978-3-642-00202-1_24.

27 Dániel Marx. Closest Substring Problems with Small Distances. *SIAM J. Comput.*, 38(4):1382–1410, 2008. doi:10.1137/060673898.

28 N. Megiddo and K. Supowit. On the Complexity of Some Common Geometric Location Problems. *SIAM J. Computing*, 13(1):182–196, 1984. doi:10.1137/0213014.

29 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and Near-Optimal Derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–191. IEEE, 1995.

30 Christian Sohler and David P. Woodruff. Strong Coresets for k-Median and Subspace Approximation: Goodbye Dimension. In *Proceedings of the 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 802–813. IEEE, 2018. doi:10.1109/FOCS.2018.00081.

Nonnegative Rank Measures and Monotone Algebraic Branching Programs

Hervé Fournier
Université de Paris, IMJ-PRG, CNRS, F-75013 Paris, France

Guillaume Malod
Université de Paris, IMJ-PRG, CNRS, F-75013 Paris, France

Maud Szusterman
Université de Paris, IMJ-PRG, CNRS, F-75013 Paris, France

Sébastien Tavenas
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000 Chambéry, France

Abstract

Inspired by Nisan's characterization of noncommutative complexity (Nisan 1991), we study different notions of nonnegative rank, associated complexity measures and their link with monotone computations. In particular we answer negatively an open question of Nisan asking whether nonnegative rank characterizes monotone noncommutative complexity for algebraic branching programs. We also prove a rather tight lower bound for the computation of elementary symmetric polynomials by algebraic branching programs in the monotone setting or, equivalently, in the homogeneous syntactically multilinear setting.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Theory of computation → Complexity classes

Keywords and phrases Elementary symmetric polynomials, lower bounds

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.15

Related Version A full version of the paper is available at https://eccc.weizmann.ac.il/report/2019/100/.

Funding *Sébastien Tavenas*: This work was partially funded by the project BIRCA from the IDEX Université Grenoble Alpes.

1 Introduction

Measures based on rank are one of the main tools to prove lower bounds in algebraic complexity theory. The complexity of first-order partial derivatives is the key ingredient for the best lower bound known for general circuits [2]. When looking at higher-order partial derivatives, one can consider their rank: the rank of partial derivatives, and some variants, have been intensively used to obtain lower bounds on restricted models [20, 21, 18]. Nisan [19] provided one of the earliest and cleanest examples of such a measure: when computing a polynomial over noncommuting variables by a so-called *algebraic branching program*, it gives an exact characterization of the complexity.[1] To state this result more precisely, let us give here the definition of algebraic branching programs used in [19].

► **Definition 1.** *An* algebraic branching program *(ABP) is a layered directed acyclic graph with a source s and a sink t. The first layer contains only the source s, the last layer contains only the sink t. Edges can only appear between vertices of successive layers and carry a weight*

[1] It was noticed in [7] (see also [17]) that this result actually follows from an older characterization for word series [10]. This characterization was also extended to tree series in [4], which can be applied to circuits.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 15; pp. 15:1–15:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

which is a linear form of the variables. The weight of a path from s to t is the product of the weights of its edges. The (homogeneous) polynomial computed by the ABP is the sum of the weights of the paths from s to t. The width of a layer is the number of vertices on that layer.

This definition makes sense both in the usual case of commuting variables and in the case of noncommuting variables, over a free algebra, which we consider for the moment. For a noncommutative homogeneous polynomial P of degree d over variables in X, define matrices $M_i(P)$, for $0 \leqslant i \leqslant d$: the rows of $M_i(P)$ are indexed by all possible monomials u over X of degree i, the columns are indexed by all possible monomials v over X of degree $d-i$, and the coefficient (u, v) of $M_i(P)$ is the coefficient of the monomial uv in P. We call this matrix the i-th Nisan matrix of P. The characterization is then expressed by the following theorem.

▶ **Theorem 2** (Nisan [19], Fliess [10]). *The size of a smallest ABP computing a noncommutative polynomial P is the sum of the ranks of its Nisan matrices, i.e., $\sum_{i=0}^{d} \operatorname{rk} M_i(P)$. More precisely, the value $\operatorname{rk} M_i(P)$ is the width of the i-th layer in a smallest ABP computing P. It is also the smallest possible width of the i-th layer in any ABP computing P.*

Nisan also considers the case of monotone noncommutative computations. In this case Nisan does not obtain a characterization of monotone noncommutative complexity, but a sufficient tool for lower bounds, using the notion of nonnegative rank.

▶ **Definition 3.** *An ABP over an ordered field is called* monotone *if all coefficients of linear forms on the edges are nonnegative.*

▶ **Definition 4.** *The* nonnegative rank *of a nonnegative matrix M, $\operatorname{rk}^+ M$, is the smallest integer r such that M can be written as a sum of r rank-1 nonnegative matrices.*

▶ **Proposition 5** (Nisan [19]). *For a polynomial P with nonnegative coefficients, the value $\operatorname{rk}^+ M_i(P)$ is the smallest possible width of the i-th layer in a monotone ABP computing P. The size of a smallest monotone ABP computing P is therefore at least $\sum_{i=0}^{d} \operatorname{rk}^+ M_i(P)$.*

Nisan [19] leaves the tightness of the inequality in Proposition 5 as an open question: does nonnegative rank also provide a characterization of monotone noncommutative complexity? One of our main results is a negative answer to this question (Theorem 25).

Before that, we consider in Section 2 a more general notion of monotone computation, which we call *weakly monotone*. Where monotonicity completely disallows cancellations, weak monotonicity allows them as long as any monomial appearing in the computation also appears in the end result. In other words, cancellations can be used to obtain the specific coefficients of a polynomial, but not to produce and then cancel out monomials outside the support of the polynomial. We strengthen Proposition 5 for weakly monotone noncommutative ABPs using a new rank measure. We then obtain a separation showing that weakly monotone noncommutative arithmetic formulas can be exponentially more powerful than monotone noncommutative ABPs. Thus weakly monotone lower bounds are stronger than monotone lower bounds.

In Section 3 we prove Theorem 25, answering Nisan's question, and more generally explore the link between nonnegative rank measures and the size of monotone noncommutative algebraic branching programs.

Finally, in Section 4 we focus on proving lower bounds for monotone *commutative* ABPs, building on ideas from the previous sections to develop new tools. Imposing monotonicity as a restriction on arithmetic computations to prove lower bounds has a long history [22, 15], which often involves hard polynomials and yields exponential lower bounds. We focus here on the elementary symmetric polynomials $e_{n,k}$. While it is known that the elementary symmetric polynomials $e_{n,k}$ require monotone, or even homogeneous multilinear, formulas of size $k^{\Omega(k)}n$ [13], these can be efficiently computed by monotone ABPs of size $O(k(n-k))$: we give a simple dynamic programming construction and further references in the full version [11]. The existence of efficient computations imply that our lower bound techniques must be very precise. Surprisingly, there is also a very simple $\Omega(k(n-k+1))$ monotone lower bound in [16], but in a model where edge weights can only be a scalar or a scalar times a variable, not linear forms, which would only give a trivial lower bound in our setting. Our second main result is a similarly quadratic lower bound for our model, and for weakly monotone computations, at the cost of a significant increase in the complexity of the proof: we use a generalization of a combinatorial problem known as Galvin's problem.[2] Our lower bound can be equivalently stated as applying to homogeneous syntactically multilinear ABPs.

Let us add one remark on the definition of ABPs. This computation model is inherently homogeneous and we only consider nonzero homogeneous polynomials. We could also consider nonhomogeneous ABPs: these are directed acyclic graphs with a source and a sink, not necessarily layered, with arcs labelled with affine forms. In the noncommutative case, when computing a homogeneous polynomial, one can show that there is always a minimal-size ABP which is homogeneous and corresponds to Definition 1. We provide a proof sketch in the full version [11]. This is also true in the commutative case for weakly monotone computations. Hence we shall consider only homogeneous branching programs.

Throughout the paper we use $\mathbb{R}$ in the case of an ordered field, but these results hold over any ordered field. When the field is not ordered we denote it by $\mathbb{K}$ and assume it is of characteristic 0.

2 A rank measure for weakly monotone computations

2.1 Weakly monotone computations

As defined before, the weight of a path is the product of the weights of its edges, i.e., a product of linear forms. Any of the monomials obtained when expanding completely this product, by choosing one term in each linear form, is said to be *produced along the path.*

▶ **Definition 6.** *An ABP is called* weakly monotone *if any monomial produced along a path from the source to the sink has a nonzero coefficient in the polynomial computed by this ABP.*

Note that this notion of monotonicity makes sense both in the commutative and noncommutative settings (Sections 2 and 3 deal with noncommutative computations, while we will use the commutative case in Section 4). We now define a new measure for weakly monotone computations. We will denote the support of a matrix M by $\operatorname{supp} M$: it is the subset of the coordinates of M which correspond to nonzero entries.

▶ **Definition 7.** *The* weakly nonnegative rank *of a matrix M, denoted by $\operatorname{rk}^{\mathrm{w}} M$, is the smallest number r such that there exist $M_1, \ldots, M_r$ of rank 1 (with entries of any sign) such that $\operatorname{supp} M_i \subseteq \operatorname{supp} M$ for all i and $\sum_{i=1}^{r} M_i = M$.*

The usual nonnegative rank of a matrix already plays a role in several areas. For instance, the fact that the minimum number of facets of an extension of a polyhedron is equal to the nonnegative rank of its so-called slack matrix. In another direction, for a $0,1$-matrix M,

[2] A different generalization of this combinatorial problem was recently used to prove almost quadratic lower bounds on the size of syntactically multilinear circuits [1].

$\log(\operatorname{rk} M)$ is a lower bound on the communication complexity of the associated problem. The *log-rank conjecture* stipulates that there is also a $\log^{O(1)}(\operatorname{rk} M)$ upper bound. This conjecture is known to be equivalent to the fact that for any $0,1$-matrix M, $\log(\operatorname{rk}^+ M) = \log^{O(1)}(\operatorname{rk} M)$. The influence of communication complexity will be felt here too, as it can be seen from the use of the support of the matrix in the definition of weakly nonnegative rank. In fact, we will borrow a few more basic concepts from communication complexity.

▶ **Definition 8.** *For a matrix M with rows indexed by a set I and columns indexed by a set J, a* combinatorial rectangle *is a subset of $I \times J$ of the form $A \times B$, with $A \subseteq I$ and $B \subseteq J$.*

A cover *of a matrix M is a set of combinatorial rectangles, included in the support of M and whose union is equal to the support of M. We define* $\operatorname{cov} M$ *as the smallest size of a cover of M.*

▶ **Proposition 9.** *We have* $\operatorname{cov} M \leqslant \operatorname{rk}^{\mathrm{w}} M$ *and* $\operatorname{rk} M \leqslant \operatorname{rk}^{\mathrm{w}} M$*. For a nonnegative matrix M,* $\operatorname{rk}^{\mathrm{w}} M \leqslant \operatorname{rk}^+ M$*.*

Let us remark that we can have $\operatorname{rk} M < \operatorname{rk}^{\mathrm{w}} M$: this is the case for the following matrix [5], for which $\operatorname{rk} R = 3$ and $\operatorname{cov} R = 4$: $R = \begin{pmatrix} 1&0&1&0\\ 0&1&0&1\\ 0&1&1&0\\ 1&0&0&1 \end{pmatrix}$.

The following proposition is the weak monotone version of Proposition 5.

▶ **Proposition 10.** *For a noncommutative polynomial P, the smallest size of the i-th layer of a weakly monotone ABP computing P is equal to* $\operatorname{rk}^{\mathrm{w}} M_i(P)$*. Hence the weakly monotone ABP size of P is greater or equal to* $\sum_i \operatorname{rk}^{\mathrm{w}} M_i(P)$*.*

Proof. Let $\ell \in \{1, \ldots, d-1\}$ and let $M = M_\ell(P)$, $r = \operatorname{rk}^{\mathrm{w}} M$.

Consider a weakly monotone ABP computing P. Let s be the size of layer ℓ. Cutting the ABP at layer ℓ we get $P = \sum_{i=1}^{s} L_i R_i$. Let A_i be the matrix of the polynomial $L_i R_i$. The matrices $A_1, \ldots, A_s$ satisfy all the conditions to show that $\operatorname{rk}^{\mathrm{w}} M \leqslant s$.

Conversely, write $M_\ell(P) = A_1 + \ldots + A_r$ where A_i are rank 1 matrices with $\operatorname{supp} A_i \subseteq \operatorname{supp} M$. Each A_i can be interpreted as a product of two polynomials $L_i R_i$. It is easy to design a weakly monotone ABP with ℓ-layer of size r computing the polynomials $L_1, \ldots, L_r$ on the ℓ-th layer.

So we have proved that for any ℓ, the minimal size of the ℓ-th layer of a weakly monotone ABP computing P is equal to $\operatorname{rk}^{\mathrm{w}} M_\ell(P)$. The last inequality follows from the fact that the minimal size of a weakly monotone ABP computing P is greater or equal to the sum of the minimal size of the different layers. ◀

2.2 Separation of rank measures

We show now that we can have $\operatorname{rk}^{\mathrm{w}} M < \operatorname{rk}^+ M$. In the following J is the matrix with all entries equal to 1 and $\|\cdot\|$ is the infinite norm.

▶ **Proposition 11.** *Let M be a nonnegative matrix. For $\varepsilon > 0$ small enough, $N = M + \varepsilon J$ satisfies* $\operatorname{rk}^{\mathrm{w}} N \leqslant \operatorname{rk} M + 1$ *and* $\operatorname{rk}^+ N \geqslant \operatorname{cov} M$*.*

Proof. We have $\operatorname{rk} N \leqslant \operatorname{rk} M + 1$ because J is of rank 1, and $\operatorname{rk} N = \operatorname{rk}^{\mathrm{w}} N$ since the support of N is full. Hence $\operatorname{rk}^{\mathrm{w}} N \leqslant \operatorname{rk} M + 1$.

It remains to show the lower bound on $r = \operatorname{rk}^+ N$. Write $N = N_1 + \ldots + N_r$ with N_i nonnegative matrix of rank 1. Write $N_i = a_i b_i^{\mathrm{T}}$ with a_i and b_i nonnegative satisfying $\|a_i\| = \|b_i\|$: this implies that $\|a_i\|, \|b_i\| \leqslant \sqrt{\|N\|} \leqslant 2\sqrt{\|M\|}$ for ε small enough. Let $\tilde{a}_i$

and $\tilde{b}_i$ be obtained from a_i and b_i by putting to 0 all entries smaller or equal to $\sqrt{\varepsilon}$. Let $\tilde{N}_i = \tilde{a}_i\tilde{b}_i^{\mathrm{T}}$. The support of $\tilde{N}_i$ is a combinatorial rectangle. Moreover, $\operatorname{supp} \tilde{N}_i \subseteq \operatorname{supp} M$ since any nonzero entry of $\tilde{N}_i$ is greater than ε.

Let us show that $\operatorname{supp} M \subseteq \bigcup_i \operatorname{supp} \tilde{N}_i$. Let $\tilde{N} = \tilde{N}_1 + \ldots + \tilde{N}_r$. For any entry (x, y), we have $|N_i(x, y) - \tilde{N}_i(x, y)| \leqslant 2\sqrt{\varepsilon}\sqrt{\|M\|}$ so $\|N - \tilde{N}\| \leqslant 2r\sqrt{\varepsilon\|M\|}$. This shows that $\|M - \tilde{N}\| \leqslant \|M - N\| + \|N - \tilde{N}\| \leqslant \varepsilon + 2r\sqrt{\varepsilon\|M\|}$. Hence $\|M - \tilde{N}\|$ is smaller than the smallest nonzero entry of M for ε small enough. This proves that $\operatorname{supp} M$ is covered by $\bigcup_{i=1}^r \operatorname{supp} \tilde{N}_i$. ◀

We want to apply the previous proposition to a matrix with a large gap between rank and covering bound. Such examples are known: the $n \times n$ matrix defined by $M_{i,j} = (a_i - a_j)^2$ for distinct reals $a_1, \ldots, a_n$ has rank 3 but $\operatorname{cov} M = \Omega(\log n)$ [3]; the slack matrix of a generic polygon also exhibits such a gap [9] (note that this matrix is not explicit).

We will build on a third construction to obtain an exponential separation between weakly monotone formulas and monotone ABPs in the noncommutative setting. Let U_n be the matrix whose rows and columns are indexed by $\{0, 1\}^n$ and which is defined by $U_n(u, v) = (\langle u, v\rangle - 1)^2$, where the scalar product is over $\mathbb{R}$.

▶ **Theorem 12** ([6], see also [8]). *It holds that* $\operatorname{rk} U_n = O(n^2)$ *and* $\operatorname{cov} U_n = 2^{\Omega(n)}$.

Using Proposition 11, this theorem gives a matrix with an exponential gap between weakly nonnegative rank and nonnegative rank.

▶ **Proposition 13.** *For* $\varepsilon > 0$ *small enough,* $\operatorname{rk}^{\mathrm{w}}(U_n + \varepsilon J) = O(n^2)$ *and* $\operatorname{rk}^{+}(U_n + \varepsilon J) = 2^{\Omega(n)}$.

2.3 Separating noncommutative monotone and weakly monotone classes

Let us define a noncommutative polynomial over the variables $\{x_0, x_1\}$. For $u \in \{0, 1\}^n$, let $x_u = x_{u_1} \ldots x_{u_n}$ and define $P = \sum_{u,v\in\{0,1\}^n} (\langle u, v\rangle - 1)^2 x_u x_v$. This polynomial was used in [14] to obtain the following separation.

▶ **Theorem 14** ([14]). *The noncommutative polynomial* P *defined above has formula size* $O(n^3)$ *but monotone circuit size* $2^{\Omega(n)}$.

As a consequence, we get a separation illustrating the difference between monotone and weakly monotone computations.

▶ **Definition 15.** *A formula is called* weakly monotone *if any monomial produced by the computation (before any possible cancellations) has a nonzero coefficient in the computed polynomial. More formally, a formula is weakly monotone if any monomial produced by a parse tree is present in the computed polynomial.*

▶ **Theorem 16.** *For* $\varepsilon > 0$ *small enough, the noncommutative polynomial* $P + \varepsilon(x_0 + x_1)^{2n}$ *has weakly monotone formula size* $O(n^3)$ *but requires monotone ABP size* $2^{\Omega(n)}$.

Proof. Let $Q = P + \varepsilon(x_0 + x_1)^{2n}$ for some $\varepsilon > 0$ small enough. The polynomial P has formula size $O(n^3)$ by the upper bound from Theorem 14. The polynomial $\varepsilon(x_0 + x_1)^{2n}$ has formula size $O(n)$. Since the support of Q is full, the formula obtained for Q by summing these two formulas is weakly monotone.

The middle Nisan matrix of Q is $M_n(Q) = U_n + \varepsilon J$ so $\operatorname{rk}^{+} M_n(Q) = 2^{\Omega(n)}$ by Proposition 13. It follows from Proposition 5 that Q has monotone ABP size $2^{\Omega(n)}$. ◀

3 Monotone noncommutative complexity vs monotone rank measures

This section is devoted to the comparison between nonnegative rank measures and the size of monotone noncommutative algebraic branching programs, in particular Nisan's question on the tightness of the lower bound for monotone noncommutative ABPs. Let us start by a particular case where the inequality is tight.

We work over a field $\mathbb{K}$ of characteristic zero. We say a vector v is a weakly monotone linear combination of $u_1, \ldots, u_p$ if there exist scalars λ_i for $1 \leqslant i \leqslant p$ such that no cancellations occur:

$$v = \sum_{i \in [1,p]} \lambda_i u_i \quad \text{with} \quad \operatorname{supp}(v) = \bigcup_{\substack{i \in [1,p] \\ \lambda_i \neq 0}} \operatorname{supp}(u_i).$$

3.1 In the case of ranks at most 2

In the case where each Nisan matrix is of rank at most 2, we prove that an algebraic branching program of minimal size can be chosen to be monotone (or weakly monotone). Since $\operatorname{rk} M \leqslant 2$ implies $\operatorname{rk} M = \operatorname{rk}^{\mathrm{w}} M = \operatorname{rk}^{+} M$, it means that the measures $\sum_i \operatorname{rk}^{+} M_i(P)$ and $\sum_i \operatorname{rk}^{\mathrm{w}} M_i(P)$ do characterize the monotone and weakly monotone ABP size in this case.

The proof of the next two lemmas can be found in the full version [11].

▶ **Lemma 17.** *If a homogeneous noncommutative polynomial P of degree d with nonnegative coefficients satisfies $\operatorname{rk} M_i(P) = 2$ for all $0 < i < d$, then there exists a monotone ABP of width 2 computing P. Hence the minimal size of a monotone ABP computing P is equal to $\sum_{i=0}^{d} \operatorname{rk}^{+} M_i(P)$.*

▶ **Lemma 18.** *If P is a homogeneous noncommutative polynomial of degree d such that $\operatorname{rk} M_i(P) = 2$ for all $0 < i < d$, there exists a weakly monotone ABP of width 2 computing P. Hence the minimal size of a weakly monotone ABP computing P is equal to $\sum_{i=0}^{d} \operatorname{rk}^{\mathrm{w}} M_i(P)$.*

Then we can easily conclude:

▶ **Theorem 19.** *Let P be a noncommutative polynomial, homogeneous of degree $d > 0$, such that $\operatorname{rk} M_i(P) \leqslant 2$ for all i. Then the minimal size of a weakly monotone ABP computing P is equal to $\sum_{i=0}^{d} \operatorname{rk}^{\mathrm{w}} M_i(P)$. Moreover, if P is nonnegative, the minimal size of a monotone ABP computing P is equal to $\sum_{i=0}^{d} \operatorname{rk}^{+} M_i(P)$.*

Proof. Assume P is nonnegative homogeneous of degree $d > 0$. We prove the second point by induction on d. If $d = 1$ the polynomial P is linear with nonnegative coefficients, $P \neq 0$, and thus $\operatorname{rk}^{+} M_0(P) + \operatorname{rk}^{+} M_1(P) = 2$, which is the size of a minimal monotone ABP. Assume now that $d > 1$. If $\operatorname{rk} M_i(P) = 2$ for all $0 < i < d$, then the minimal size of a monotone ABP computing P is equal to $\sum_{i=0}^{d} \operatorname{rk}^{+} M_i(P)$ by Lemma 17. Otherwise, there exists $0 < i < d$ such that $\operatorname{rk}^{+} M_i(P) = 1$. It means that $P = QR$ with Q and R homogeneous of degree i and $d - i$. By induction the minimal size of a monotone ABP computing Q is equal to $\sum_{i=0}^{d} \operatorname{rk}^{+} M_i(Q)$ and similarly for R. The conclusion follows easily for P.

The proof of the first point is analogous, using Lemma 18. ◀

3.2 Separation of monotone rank measure and ABP size

We now prove a separation between the sum-of-ranks measure and the minimal noncommutative ABP size, both in the monotone and in the weakly monotone cases.

If $X = X_1 \uplus \ldots \uplus X_d$ is a partition of the set of variables, a noncommutative polynomial f is called *ordered* over the family $X_1, \ldots, X_d$ if it is homogeneous of degree d and if each monomial m from f is of the form $v_1 v_2 \cdots v_d$, where $v_i \in X_i$ for each i.

We will use several times the following very simple observation.

▶ **Proposition 20.** *Let $\mathcal{E}_i$ be the i-th coordinate hyperplane of $\mathbb{K}^n$. and $\mathcal{E} = \bigcap_{i \in I} \mathcal{E}_i$ for $I \subseteq [n]$. Let $v \in \mathcal{E}$, and $u_1, \ldots, u_p \in \mathbb{K}^n$. Assume $v = \sum_{j=1}^{p} \lambda_j u_j$ is a weakly monotone linear combination. Then for all j such that $\lambda_j \neq 0$, we have $u_j \in \mathcal{E}$.*

In particular, if $v \neq 0$ is a weakly monotone linear combination $v = \sum_j \lambda_j u_j$, then there exists j_0 such that $\lambda_{j_0} \neq 0$ and $u_{j_0} \in \mathcal{E} \setminus \{0\}$.

▶ **Lemma 21.** *There exists a noncommutative ordered degree 3 polynomial H with nonnegative coefficients in $\mathbb{R}$ over the set of variables (X, Y, Z) with $|X| = 4, |Y| = 2, |Z| = 4$, such that $\operatorname{rk}^+ M_i(H) = \operatorname{rk}^{\mathrm{w}} M_i(H) = \operatorname{rk} M_i(H) = 3$ for $i \in \{1, 2\}$, so that $\sum_{i=0}^{3} \operatorname{rk}^+ M_i(H) = \sum_{i=0}^{3} \operatorname{rk}^{\mathrm{w}} M_i(H) = 8$, but the minimal size of a monotone ABP and of a weakly monotone ABP is 9.*

Proof. Define the vectors $A = \begin{pmatrix} 1\\0\\0\\1 \end{pmatrix}, B = \begin{pmatrix} 0\\1\\1\\0 \end{pmatrix}, C = \begin{pmatrix} 1\\0\\1\\0 \end{pmatrix}, D = \begin{pmatrix} 0\\1\\0\\1 \end{pmatrix}$ (they correspond to the columns of the matrix R of Section 2). Then, $\operatorname{rk}(A, B, C, D) = 3$ and $\operatorname{rk}^{\mathrm{w}}(A, B, C, D) = 4$, since $\operatorname{cov}(A, B, C, D) = 4$. Define the matrices $M \in \mathbb{R}_{\geqslant 0}^{4 \times 8}$ and $N \in \mathbb{R}_{\geqslant 0}^{8 \times 4}$: $M \stackrel{\text{def}}{=} \begin{pmatrix} A & B & \frac{A+C}{2} & \frac{B+C}{2} & C & C & \frac{C+D}{2} & \frac{C+D}{2} \end{pmatrix}$ and $N \stackrel{\text{def}}{=} \begin{pmatrix} A & B & \frac{A+C}{2} & \frac{B+C}{2} \\ C & C & \frac{C+D}{2} & \frac{C+D}{2} \end{pmatrix}$. As $\frac{C+D}{2} = \frac{A+B}{2}$, the columns of M are monotone linear combinations of A, B and C. Moreover, the columns of N are monotone linear combinations of $\binom{A}{C}, \binom{B}{C}$ and $\binom{C}{D}$. Hence, $\operatorname{rk}^+ M = \operatorname{rk}^+ N = 3$. This shows that $\operatorname{rk}^{\mathrm{w}} M = \operatorname{rk}^{\mathrm{w}} N = 3$.

Let $X = \{x_1, x_2, x_3, x_4\}$, $Y = \{y_1, y_2\}$ and $Z = \{z_1, z_2, z_3, z_4\}$. We consider the ordered polynomial $H \in \mathbb{R}_{\geqslant 0}[X, Y, Z]$:

$$\begin{aligned} H \stackrel{\text{def}}{=}\ & x_1y_1z_1 + x_4y_1z_1 + x_2y_1z_2 + x_3y_1z_2 + x_1y_1z_3 + \frac{1}{2}x_3y_1z_3 + \frac{1}{2}x_4y_1z_3 + \frac{1}{2}x_1y_1z_4 \\ & + \frac{1}{2}x_2y_1z_4 + x_3y_1z_4 + x_1y_2z_1 + x_3y_2z_1 + x_1y_2z_2 + x_3y_2z_2 + \frac{1}{2}x_1y_2z_3 + \frac{1}{2}x_2y_2z_3 \\ & + \frac{1}{2}x_3y_2z_3 + \frac{1}{2}x_4y_2z_3 + \frac{1}{2}x_1y_2z_4 + \frac{1}{2}x_2y_2z_4 + \frac{1}{2}x_3y_2z_4 + \frac{1}{2}x_4y_2z_4. \end{aligned}$$

One can verify than the middle Nisan matrices of H are $M_1(H) = M$ and $M_2(H) = N$.

Assume that there exists a weakly monotone noncommutative homogeneous ABP of size $8 = \sum \operatorname{rk}^{\mathrm{w}} M_i(H)$ computing H. It means that the ABP has exactly $\operatorname{rk}^{\mathrm{w}} M_i$ nodes at layer i for $0 \leqslant i \leqslant 3$. In particular, such an ABP has three nodes at layer 1, each one computing a polynomial $P_1^{(1)}(X)$, $P_2^{(1)}(X)$ and $P_3^{(1)}(X)$ and has also three nodes at layer 2 which compute the polynomials $P_1^{(2)}(X, Y)$, $P_2^{(2)}(X, Y)$ and $P_3^{(2)}(X, Y)$. The goal is to show that these triplets of polynomials are precisely defined and there is no way to link them together in a weakly monotone ABP. By definition of the Nisan matrix, we can see columns of M as polynomials in $\mathbb{R}[X]$ and columns of N as polynomials in $\mathbb{R}[X, Y]$.

▷ Claim 22. The polynomials $P_1^{(1)}$, $P_2^{(1)}$ and $P_3^{(1)}$ weakly monotonically generate the columns of M and the polynomials $P_1^{(2)}$, $P_2^{(2)}$ and $P_3^{(2)}$ weakly monotonically generate the columns of N.

Proof. Let us show the result at layer 1, the case of layer 2 is symmetrical. Consider a column C of the first Nisan matrix: say it corresponds to the coefficient of $y_j z_k$ in H. If we instantiate the variables y_j and z_k to 1 and the other variables from $Y \cup Z$ to 0 in the ABP, we get C as a linear combination of columns representing $P_1^{(1)}(X)$, $P_2^{(1)}(X)$ and $P_3^{(1)}(X)$. More precisely, $C = \sum_{s=1}^{3} \lambda_s P_s^{(1)}$ where $\lambda_s \neq 0$ if and only if we can read the monomial $y_j z_k$ between the node corresponding to $P_s^{(1)}(X)$ and the output of the ABP.

It remains to show that this linear combination $C = \sum_{s=1}^{3} \lambda_s P_s^{(1)}$ is weakly monotone. Assume this is not, it means for some i the coefficient of x_i is 0 in C but there exists s such that $\lambda_s \neq 0$ and the coefficient of x_i in $P_s^{(1)}$ is different to 0. It means that the coefficient of $x_i y_j z_k$ is 0 in H but there is a path in the ABP with nonzero coefficient for this monomial (otherwise the scalar in front of $P_s^{(1)}(X)$ would be 0). It contradicts the fact that the ABP is weakly monotone. $\lhd$

$\rhd$ Claim 23. If three vectors U, V and W weakly monotonically generate the family (A, B, C) then (up-to permuting the names of U, V and W), $U \in \mathbb{R}A$,$V \in \mathbb{R}B$ and $W \in \mathbb{R}C$.

Proof. As $\operatorname{rk}(A, B, C) = 3$, we can consider the vector-space $\mathcal{F}$ generated by $\{A, B, C\}$, namely $\mathcal{F} = \{T \in \mathbb{R}^4 \mid t_1 + t_2 - t_3 - t_4 = 0\}$. So the vectors U, V and W must form a basis of $\mathcal{F}$ and so, have to lie in $\mathcal{F}$. For $1 \leqslant i \leqslant 4$, let $\mathcal{E}_i$ be the i-th coordinate hyperplane of $\mathbb{R}^4$.

Notice that $\mathbb{R}A = \mathcal{F} \cap \mathcal{E}_2 \cap \mathcal{E}_3$. By Proposition 20, since A is a weakly monotone linear combination of U, V, W, (at least) one of the vectors $\{U, V, W\}$ must belong to $\mathcal{E}_2 \cap \mathcal{E}_3$. Since this vector lies also in $\mathcal{F}$, it is in $\mathbb{R}A$.

In the same way, since $\mathbb{R}B = \mathcal{F} \cap \mathcal{E}_1 \cap \mathcal{E}_4$ and $\mathbb{R}C = \mathcal{F} \cap \mathcal{E}_2 \cap \mathcal{E}_4$, one vector of $\mathcal{B}$ must belong to $\mathbb{R}B$ and one must belong to $\mathbb{R}C$.

Since $\mathbb{R}A, \mathbb{R}B, \mathbb{R}C$ are 3 distinct one-dimensional linear subspaces, each one of these spaces has to contain one of the vectors U, V, W. $\lhd$

$\rhd$ Claim 24. If three vectors Q, R and S weakly monotonically generate the columns of N then, up to permuting the names of Q, R and S, $Q \in \mathbb{R}\left(\begin{smallmatrix} A \\ C \end{smallmatrix}\right), R \in \mathbb{R}\left(\begin{smallmatrix} B \\ C \end{smallmatrix}\right), S \in \mathbb{R}\left(\begin{smallmatrix} C \\ D \end{smallmatrix}\right)$.

Proof. Let us define $\mathcal{B} = \{Q, R, S\}$. We can easily see that the columns of N lie in the vector space given by the intersection of the three hyperplanes $\mathcal{F}_1 = \{T \in \mathbb{R}^8 \mid t_1 + t_2 = t_3 + t_4\}$, $\mathcal{F}_2 = \{T \in \mathbb{R}^8 \mid t_5 + t_6 = t_7 + t_8\}$ and $\mathcal{F}_3 = \left\{T \in \mathbb{R}^8 \mid \sum_{i=1}^{4} t_i = \sum_{j=5}^{8} t_j\right\}$. As $\operatorname{rk}(Q, R, S) = \operatorname{rk} N$, the vectors Q, R and S are in $\mathcal{F}_1 \cap \mathcal{F}_2 \cap \mathcal{F}_3$.

For $1 \leqslant i \leqslant 8$, we denote by $\mathcal{E}_i$ the i-th coordinate hyperplane of $\mathbb{R}^8$. Looking at the two first columns of N we can notice that $\mathbb{R}\left(\begin{smallmatrix} A \\ C \end{smallmatrix}\right) = \mathcal{F}_1 \cap \mathcal{F}_2 \cap \mathcal{F}_3 \cap \mathcal{E}_2 \cap \mathcal{E}_3 \cap \mathcal{E}_6 \cap \mathcal{E}_8$ and $\mathbb{R}\left(\begin{smallmatrix} B \\ C \end{smallmatrix}\right) = \mathcal{F}_1 \cap \mathcal{F}_2 \cap \mathcal{F}_3 \cap \mathcal{E}_1 \cap \mathcal{E}_4 \cap \mathcal{E}_6 \cap \mathcal{E}_8$ are two distinct one-dimensional spaces of $\mathbb{R}^8$. By Proposition 20, there is at least one vector of $\mathcal{B}$ (let us suppose this is Q) such that $Q = q\left(\begin{smallmatrix} A \\ C \end{smallmatrix}\right)$ and at least one other vector of $\mathcal{B}$ (assume this is R) such that $R = r\left(\begin{smallmatrix} B \\ C \end{smallmatrix}\right)$.

Finally we need to identify the last vector S. For that, we decompose $S = \left(\begin{smallmatrix} S_1 \\ S_2 \end{smallmatrix}\right)$ where S_1 is the projection of S on its first four coordinates and S_2 the projection on the last four. Now, we know that $\mathcal{B}$ weakly monotonically generates the last two columns of N. So there exist $a_1, a_2, a_3, b_1, b_2, b_3 \in \mathbb{R}$ such that: $a_1 q\left(\begin{smallmatrix} A \\ C \end{smallmatrix}\right) + a_2 r\left(\begin{smallmatrix} B \\ C \end{smallmatrix}\right) + a_3\left(\begin{smallmatrix} S_1 \\ S_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} \frac{A+C}{2} \\ \frac{C+D}{2} \end{smallmatrix}\right)$ and $b_1 q\left(\begin{smallmatrix} A \\ C \end{smallmatrix}\right) + b_2 r\left(\begin{smallmatrix} B \\ C \end{smallmatrix}\right) + b_3\left(\begin{smallmatrix} S_1 \\ S_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} \frac{B+C}{2} \\ \frac{C+D}{2} \end{smallmatrix}\right)$. By Proposition 20, as $\left(\begin{smallmatrix} \frac{A+C}{2} \\ \frac{C+D}{2} \end{smallmatrix}\right) \in \mathcal{E}_2$ and $\left(\begin{smallmatrix} \frac{B+C}{2} \\ \frac{C+D}{2} \end{smallmatrix}\right) \in \mathcal{E}_4$, we know that $a_2 = b_1 = 0$. Moreover, as A and $(A + C)$ are not colinear, it means that S_1 belongs to the plane $\operatorname{vect}(A, A + C)$ (by the way, this space is inside $\mathcal{E}_2$). Similarly, $S_1 \in \operatorname{vect}(B, B + C)$. As $B \notin \mathcal{E}_2$, these two planes are distinct, so the intersection is of dimension at most 1. Moreover, $\operatorname{vect}(C)$ is in the intersection, and so, $S_1 \in \operatorname{vect}(C)$. There exists $s \neq 0$ such that $S_1 = sC$. As $a_1 q A + a_3 s C = \frac{A+C}{2}$, it implies that $a_1 q = a_3 s = \frac{1}{2}$. Then, we have $\frac{C}{2} + \frac{S_2}{2s} = \frac{C+D}{2}$, i.e., $S_2 = sD$. $\lhd$

Consequently, by Claim 22 and Claim 24, one node at layer 2 computes the polynomial whose matrix is $s\left(\begin{smallmatrix} C \\ D \end{smallmatrix}\right)$ (with $s \neq 0$). By instantiating y_1 to 0 and y_2 to $1/s$, this node computes exactly the polynomial corresponding to D as a weakly monotone linear combination of the nodes at layer 1. By Claim 23, the nodes at layer 1 are polynomials associated to A, B and C (up to scalar multiplication). This would imply that $\mathrm{rk}^{\mathrm{w}}(A, B, C, D) = 3$, which is false. Hence, there does not exist a weakly monotone ABP of size 8.

To complete the proof, we show there is a monotone ABP of size 9 computing H. There are two natural monotone ABPs of size 9, let us describe one of them. One can compute the four polynomials associated to A, B, C and D at the first layer. It gives the following monotone ABP of size 9: $H = \frac{1}{2}\left(\begin{smallmatrix} x_1+x_4 & x_2+x_3 & x_1+x_3 & x_2+x_4 \end{smallmatrix}\right)\left(\begin{smallmatrix} y_1 & 0 & 0 \\ 0 & y_1 & 0 \\ y_2 & y_2 & y_1 \\ 0 & 0 & y_2 \end{smallmatrix}\right)\left(\begin{smallmatrix} 2z_1+z_3 \\ 2z_2+z_4 \\ z_3+z_4 \end{smallmatrix}\right)$. ◀

▶ **Theorem 25.** *There exists a noncommutative homogeneous degree 3 polynomial P over 4 variables such that $\mathrm{rk}^{+} M_i(P) = \mathrm{rk}^{\mathrm{w}} M_i(P) = \mathrm{rk}\, M_i(P) = 3$ for $i \in \{1, 2\}$, so that $\sum_{i=0}^{3} \mathrm{rk}^{+} M_i(P) = \sum_{i=0}^{3} \mathrm{rk}^{\mathrm{w}} M_i(P) = 8$, but the minimal size of a weakly monotone or monotone ABP computing P is 9.*

Proof. Consider the noncommutative polynomial $P = H(x_1, x_2, x_3, x_4, x_1, x_2, x_1, x_2, x_3, x_4)$. As H is ordered, and as the previous substitution follows this order, it is injective over the set of monomials which appear in H, that is to say, if m_1 and m_2 are two monomials from H which give the same monomial in P, then $m_1 = m_2$. It directly implies that the substitution establishes a bijection between the set of monomials which appear in H and the ones which appear in P. We will say that this substitution is faithful.

Any ABP A which computes the polynomial H can be transformed into an ABP B which computes P with layers of same size by a direct substitution of the variables. Moreover, if A is monotone, then B is immediately monotone. Then, if A is weakly monotone, the faithfulness property implies that B is also weakly monotone.

In the other direction, if in a weakly monotone noncommutative ABP A computing P we replace the variables x_1 and x_2 in the second layer by y_1 and y_2 and the variables x_1, x_2, x_3 and x_4 in the third layer by z_1, z_2, z_3 and z_4, then we get a new ABP B which computes the polynomial $H(x_1, x_2, x_3, x_4, y_1, y_2, z_1, z_2, z_3, z_4)$. The fact that this transformation preserves the monotonicity is still immediate. The faithfulness property implies it also preserves the weak monotonicity. So, the theorem follows from Lemma 21. ◀

▶ **Corollary 26** (Gap increasing with the degree and the number of variables)**.** *Let P be the polynomial defined in Theorem 25. Let $m, n \geqslant 1$. Let $X_1, \dots, X_n$ be n sets of distinct variables, with each set of size 4. Let $Q(X_1, \dots, X_n) = \sum_{j=1}^{n} P^m(X_j)$. This is a polynomial of degree $3m$ in $4n$ variables such that $\sum_{i=0}^{3m} \mathrm{rk}^{+} M_i(Q) = \sum_{i=0}^{3m} \mathrm{rk}^{\mathrm{w}} M_i(Q) = 7mn - n + 2$ but the minimal size of a monotone or weakly monotone ABP for it is equal to $8mn - n + 2$.*

Proof. Let us first consider the case $n = 1$. From Theorem 25, one can easily check that $\mathrm{rk}\, M_i(P^m(X_1)) = 1$ for i multiple of 3 and $\mathrm{rk}\, M_i(P^m(X_1)) = \mathrm{rk}^{+} M_i(P^m(X_1)) = 3$ otherwise, and that a minimal (weakly) monotone ABP computing $P^m(X_1)$ has $8m + 1$ nodes.

Consider a weakly monotone ABP for Q. Assume there is an internal node α and two distinct indices k and k' such that α depends on at least one variable of X_k and one variable of $X_{k'}$. Consequently, one path of the ABP produces a monomial which contains both a variable in X_k and a variable not in X_k. Since $Q = \sum_{j=1}^{n} P^m(X_j)$, a given monomial in Q can only contain variables coming from a single X_k. The above statement thus contradicts the fact that the ABP is weakly monotone. Hence, we can partition the internal nodes of the ABP into n parts, each one related to one variable set X_j. As mentioned earlier, a minimal weakly monotone ABP for $P^m(X_j)$ has $8m - 1$ internal nodes. The minimal size of a weakly monotone ABP is therefore $8mn - n + 2$. The same is true of a monotone ABP computing Q.

Let us compute the sum of ranks for Q. If $0 < i < 3m$, the i-th Nisan matrix of Q is block-diagonal with n blocks, where the j-th block corresponds to the i-th Nisan matrix of $P^m(X_j)$. As the nonnegative rank of a block-diagonal matrix is equal to the the sum of the nonnegative ranks of its blocks, $\operatorname{rk} M_i(Q) = \operatorname{rk}^+ M_i(Q) = n$ for $i \in \{3, 6, \dots, 3m-3\}$ and $\operatorname{rk} M_i(Q) = \operatorname{rk}^+ M_i(Q) = 3n$ for $i \equiv 1, 2 \mod 3$. Summing over the different layers we get that the sum-of-ranks measure for Q, both for usual rank and nonnegative rank, and thus for weakly nonnegative rank, is equal to $7mn - n + 2$. ◀

An upper bound on the size of a monotone ABP computing a homogeneous degree d polynomial P is obtained by summing, for each $\ell \in \{0, \dots, d\}$ the minimal number of rows extracted from $M_\ell(P)$ whose cone contains all other columns of $M_\ell(P)$. The example above shows that this is not a characterization of monotone size: for the polynomial H built in Lemma 21, it is needed to extract 4 rows in both $M_1(H)$ and $M_2(H)$. The same remark applies in the weakly monotone setting (about the minimum number of extracted rows which weakly monotonically generate all the rows).

4 Lower bounds for monotone commutative ABPs

4.1 Lower bound tools for monotone and weakly monotone ABPs

Consider a homogeneous degree d commutative polynomial P. For $\ell \in \{0, \dots, d\}$, we define the set $\mathcal{M}_\ell(P)$ of matrices, whose rows are indexed by commutative degree-ℓ monomials and whose columns indexed by degree-$(d-\ell)$ commutative monomials. A matrix M belongs to $\mathcal{M}_\ell(P)$ if:

(a) for any degree d commutative monomial m such that m does not appear in P and any (m_1, m_2) satisfying $m = m_1 m_2$, m_1 of degree ℓ and m_2 of degree $d - \ell$, we have $M_{m_1, m_2} = 0$;

(b) for any other degree d commutative monomial m, $\sum_{m_1 m_2 = m} M_{m_1, m_2}$ is equal to the coefficient of m in P.

For a matrix M whose rows and columns are indexed by noncommutative monomials, we define M^{com} the matrix obtained by summing rows and columns indexed by the same *commutative* monomial.

▶ **Proposition 27.** *A homogeneous degree-d noncommutative polynomial Q computes commutatively P without cancelling monomials if and only if $M_\ell(Q)^{com} \in \mathcal{M}_\ell(P)$ for all $\ell \in \{0, \dots, d\}$.*

Proof. The polynomial Q computes commutatively P if and only if, for each ℓ, the matrix $M := M_\ell(Q)^{com}$ satisfies the following: for any degree d commutative monomial m, $\sum_{m_1 m_2 = m} M_{m_1, m_2}$ is equal to the coefficient of m in P.

The polynomial Q does not cancel monomials if and only if, for all monomial m not appearing in P and for all decomposition $m = m_1 m_2$, there is no noncommutative monomial $m' = m'_1 m'_2$ in Q such that m'_i computes commutatively m_i for $i \in \{1, 2\}$.

Together, these two statements prove the proposition. ◀

For a homogeneous degree d polynomial P and $\ell \in \{0, \dots, d\}$ consider the support matrix $S_\ell(P)$ indexed by degree-ℓ commutative monomials on the rows, degree-$(d-\ell)$ commutative monomials on the column, such that $S_\ell(P)_{m_1, m_2} = 1$ if the coefficient of $m_1 m_2$ in P is nonzero and $S_\ell(P)_{m_1, m_2} = 0$ otherwise.

▶ **Definition 28.** *For M, S two matrices of the same size we define $\mathrm{rk}^{\mathrm{w}}(M, S)$ to be the smallest r such that there exist rank 1 matrices $M_1, \ldots, M_r$ such that $\mathrm{supp}(M_i) \subseteq \mathrm{supp}(S)$ and $M = \sum_{i=1}^{r} M_i$.*

Notice that $\mathrm{rk}^{\mathrm{w}} M$, defined in Section 2, is nothing but $\mathrm{rk}^{\mathrm{w}}(M, M)$.

▶ **Theorem 29.** *The size of a monotone ABP computing a homogeneous commutative polynomial P of degree d is at least $\sum_{\ell=0}^{d} \min\{\mathrm{rk}^{+} M \mid M \in \mathcal{M}_\ell(P),\ M \geqslant 0\}$. If the ABP is weakly monotone the bound becomes $\sum_{\ell=0}^{d} \min\{\mathrm{rk}^{\mathrm{w}}(M, S_\ell(P)) \mid M \in \mathcal{M}_\ell(P)\}$.*

Proof. Let $\ell \in \{1, \ldots, d-1\}$. Consider an ABP computing P with minimal number of nodes at level ℓ: say it is w. Cutting this ABP at layer ℓ gives a decomposition $P = \sum_{i=1}^{w} Q_i R_i$. For $i \in \{1, \ldots, w\}$ let M_i be the matrix of $Q_i R_i$. All matrices M_i are of rank 1 and we have $\sum_{i=1}^{w} M_i \in \mathcal{M}_\ell(P)$. If the ABP is monotone, the matrices M_i are nonnegative and we get $\min\{\mathrm{rk}^{+} M \mid M \in \mathcal{M}_\ell(P),\ M \geqslant 0\} \leqslant w$. If the ABP is weakly monotone, we have $\mathrm{supp}(M_i) \subseteq \mathrm{supp}(S_\ell(P))$. Hence $\min\{\mathrm{rk}^{\mathrm{w}}(M, S_\ell(P)) \mid M \in \mathcal{M}_\ell(P)\} \leqslant w$. ◀

For two same-sized matrices M, S, let $\mathrm{cov}(M, S)$ be the smallest number of combinatorial rectangles included in the support of S and whose union covers the support of M.

▶ **Proposition 30.** $\mathrm{cov}(M, S) \leqslant \mathrm{rk}^{\mathrm{w}}(M, S)$.

Proof. Let $r = \mathrm{rk}^{\mathrm{w}}(M, S)$ and write $M = \sum_{i=1}^{r} M_i$ with M_i of rank 1, $\mathrm{supp}(M_i) \subseteq \mathrm{supp}(S)$. We have $\mathrm{supp}(M) \subseteq \bigcup_{i=1}^{r} \mathrm{supp}(M_i)$: this shows that $\mathrm{cov}(M, S) \leqslant r$. ◀

▶ **Corollary 31.** *Any weakly monotone ABP computing P has size greater or equal to $\sum_{\ell=0}^{d} \min\{\mathrm{cov}(M, S_\ell(P)) \mid M \in \mathcal{M}_\ell(P)\}$.*

4.2 Application to the elementary symmetric polynomials

For n positive integer we write $[n] = \{1, \ldots, n\}$. For $0 \leqslant k \leqslant n$, let $e_{n,k}$ be the elementary symmetric polynomial of degree k over the variables $x_1, \ldots, x_n$: $e_{n,k} = \sum_{I \in \binom{[n]}{k}} \prod_{i \in I} x_i$. Notice that $S_j(e_{n,k})$ is exactly the disjointness matrix $D_{n,j,k-j}$ with rows indexed by elements of $\binom{[n]}{j}$ and columns indexed by elements of $\binom{[n]}{k-j}$, and whose entry in row A and column B is 1 if $A \cap B = \emptyset$ and 0 otherwise.

To get lower bounds for $e_{n,k}$ using Corollary 31 we need to show that, for enough values of j and for any $M \in \mathcal{M}_j(e_{n,k})$, $\mathrm{cov}(M, D_{n,j,k-j})$ is large.

▶ **Proposition 32.** *For n, j, k fixed, assume $\mathrm{cov}(M, D_{n,j,k-j}) \leqslant m$ for some $M \in \mathcal{M}_j(e_{n,k})$. Then there exists $A_1, \ldots, A_m \subseteq [n]$ with the following property:*

$$\text{For all } B \in \binom{[n]}{k},\ \text{there is } i \in \{1, \ldots, m\} \text{ such that } |A_i \cap B| = j. \qquad (1)$$

Proof. Let $M \in \mathcal{M}_j(e_{n,k})$. Assume $U_1 \times V_1, \ldots, U_m \times V_m$ is a set of combinatorial rectangles from $\binom{[n]}{j} \times \binom{[n]}{k-j}$ included in the support of $D_{n,j,k-j}$ and covering $\mathrm{supp}\, M$. Notice that such a combinatorial rectangle $U \times V$ is included in the support of $D_{n,j,k-j}$ if and only if $\left(\bigcup_{u \in U} u\right) \cap \left(\bigcup_{v \in V} v\right) = \emptyset$. For $i \in \{1, \ldots, m\}$, let $A_i = \bigcup_{u \in U_i} u$. From the previous remark the set of combinatorial rectangles $R_1, \ldots, R_m$ defined by $R_i = \binom{A_i}{j} \times \binom{[n] \setminus A_i}{k-j}$ is included in the support of $D_{n,j,k-j}$ and covers $\mathrm{supp}\, M$.

Let us show that the family $\{A_1, \ldots, A_m\}$ satisfies Equation (1). Let $B \in \binom{[n]}{k}$. The monomial $\prod_{i \in B} x_i$ appears in $e_{n,k}$ so one non-zero entry of M is of the form (I, J) with $I \in \binom{[n]}{j}$, $J \in \binom{[n]}{k-j}$ and $I \cup J = B$. Therefore $(I, J) \in R_i$ for some $i \in \{1, \ldots, m\}$, i.e. $|A_i \cap B| = |I| = j$. ◀

We will now relate our lower bound endeavor to a combinatorial question known as Galvin's problem: for n a multiple of 4, prove a lower bound on the size m of a family $\{A_1, \ldots, A_m\} \subseteq \binom{[n]}{n/2}$ such that for any $B \in \binom{[n]}{n/2}$, there exists i such that $|A_i \cap B| = n/4$. Proving a lower bound on a family $\{A_1, \ldots, A_m\}$ satisfying Equation (1) for the parameters $k = n/2$ and $j = n/4$ is a generalization of Galvin's problem because the sets A_i can be of arbitrary size, instead of $n/2$ in the original problem.

We first give a lower bound for the middle elementary symmetric polynomial. The argument is similar to the solution of Galvin's original problem presented in [12, Theorem 11.1], which we reproduce here for completeness. It is based on the following result, restricted here to the case of codes over an alphabet with 2 elements (we denote by Δ the symmetric difference between two sets).

▶ **Theorem 33** ([12], Theorem 1.10). *Suppose $0 < \delta < \frac{1}{2}$ is given. Then there exists $\varepsilon > 0$ such that for any d even satisfying $\delta n < d < (1 - \delta)n$, any family of distinct subsets $C_1, \ldots, C_m \subseteq [n]$ such that, for all $i \neq j$, $|C_i \Delta C_j| \neq d$, has size $m \leqslant (2 - \varepsilon)^n$.*

▶ **Lemma 34.** *There exists $\alpha > 0$ such that for $n \in 4\mathbb{N} \setminus \{0\}$, $k = n/2$ and j odd, any family $\{A_1, \ldots, A_m\}$ satisfying Equation (1) has size $m \geqslant \alpha n$.*

Proof. Assume there exists $A_1, \ldots, A_m \subseteq [n]$ such that $\mathcal{F} = \{A_1, \ldots, A_m\}$ satisfies Equation (1). Let V be the subspace of $\mathbb{F}_2^n$ spanned by the characteristic vectors of the elements of $\mathcal{F}$. By assumption, for all $B \in \binom{[n]}{n/2}$, there exists $F \in \mathcal{F}$ such that $|B \cap F| = j$; this means that $\langle \chi(B), \chi(F) \rangle = 1 \neq 0$ because j is odd. Hence $V^\perp$ contains no vector of weight $n/2$. Because $V^\perp$ is a vector space, it implies that for any $C, D \subseteq [n]$ such that $\chi(C), \chi(D) \in V^\perp$, $|C \Delta D| \neq n/2$.

By Theorem 33, $|V^\perp| \leqslant (2 - \varepsilon)^n$ for some constant $\varepsilon > 0$. This means that $\dim V^\perp \leqslant (1 - \alpha)n$ for some $\alpha > 0$. It follows that $m = |\mathcal{F}| \geqslant \dim V \geqslant \alpha n$. ◀

▶ **Lemma 35.** *For $n \in 4\mathbb{N}$, every weakly monotone ABP computing $e_{n,n/2}$ has size $\Omega(n^2)$.*

Proof. There exists $\alpha > 0$ such that for $n \in 4\mathbb{N}$, $k = n/2$ and j odd, any family $\{A_1, \ldots, A_m\}$ satisfying Equation (1) has size $m \geqslant \alpha n$ by Lemma 34. It follows from Proposition 32 that for all $M \in \mathcal{M}_j(e_{n,n/2})$, $\operatorname{cov}(M, D_{n,j,n/2-j}) \geqslant \alpha n$. The lower bound is obtained by Corollary 31. ◀

From the simple observation $e_{n,k}(x_1, \ldots, x_m, 0, \ldots, 0) = e_{m,k}(x_1, \ldots, x_m)$, Lemma 35 yields quadratic lower bounds on the size of weakly monotone ABPs computing $e_{n,k}$ for $\delta n \leqslant k \leqslant n/2$ for a fixed $\delta > 0$. However we need to be more careful to get a quadratic lower bound for e.g. $e_{n,2n/3}$. Indeed the simple reduction $e_{n,k}(x_1, \ldots, x_n) = \prod_{i=1}^n x_i \cdot e_{n,n-k}\left(\frac{1}{x_1}, \ldots, \frac{1}{x_n}\right)$ uses divisions, which are not allowed in our model and would cost too much to remove.

In an ABP, the formal degree $\operatorname{fdeg}_t(\alpha)$ of a node α with respect to a variable t is defined as the maximum degree in t of the polynomial computed along a path from the source to α, which is also the maximal degree in t of a monomial produced along a path from the source to α. By definition, the formal degree of the source is 0. Let us denote by $\hat{\alpha}$ the polynomial computed at the node α. Remark that $\operatorname{fdeg}_t(\alpha) \geqslant \deg_t(\hat{\alpha})$. The formal degree in t of an ABP is the formal degree in t of its output.

Let us show now that we can always extract the part of maximal formal degree without changing the size of an ABP. We denote by $[t^k]f$ the coefficient of the homogeneous component of f of degree k in t. The proofs of the next two lemmas can be found in the full version [11].

▶ **Lemma 36.** *Let A be an ABP of size s and of formal degree k in the variable t computing a polynomial f. Then there exists A' an ABP of size at most s such that A' computes $[t^k]f$.*

Moreover, if A is weakly monotone, then it is also the case for A'.

▶ **Lemma 37.** *If there is a weakly monotone ABP of size s computing the polynomial $e_{n,p}$, then for all $q \leqslant p$, there is a weakly monotone ABP of size at most s which computes the polynomial $e_{n-q,p-q}$.*

▶ **Theorem 38.** *Every weakly monotone ABP, or equivalently every homogeneous syntactically multilinear ABP, computing $e_{n,k}$ has size $\Omega(\min\{k^2, (n-k)^2\})$.*

Proof. Let us first prove the lower bound when n and k are even.

If $k \leqslant n/2$, then as mentioned previously, any weakly monotone ABP of size s implies a weakly monotone ABP of size at most s for $e_{2k,k}$ by putting some variables to 0. So in this case $s = \Omega(k^2)$ by Lemma 35.

Otherwise, we have $k > n/2$. It means that $k \geqslant 2k - n > 0$. Then a weakly monotone ABP of size s for $e_{n,k}$ gives a weakly monotone ABP of size at most s for $e_{2n-2k,n-k}$ by Lemma 37, choosing the parameters $p = k$ and $q = 2k - n$. The lower bound $s = \Omega((n-k)^2)$ follows from Lemma 35.

The lower bound is obtained for n odd by noticing that $e_{2\lfloor n/2 \rfloor,k}$ can be reduced to $e_{n,k}$ by putting one variable to 0. Moreover, $e_{n,k}$ reduces to $e_{n-1,k-1}$ by Lemma 37. So the lower bound holds for n and k of any parity.

This lower bound also holds in the homogeneous syntactically multilinear model: indeed, any such ABP computing $e_{n,k}$ is weakly monotone because $e_{n,k}$ has all degree k monomials in its support. ◀

References

1 Noga Alon, Mrinal Kumar, and Ben Lee Volk. Unbalancing Sets and an Almost Quadratic Lower Bound for Syntactically Multilinear Arithmetic Circuits. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 11:1–11:16, 2018. doi:10.4230/LIPIcs.CCC.2018.11.

2 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

3 LeRoy B. Beasley and Thomas J. Laffey. Real rank versus nonnegative rank. *Linear Algebra Appl.*, 431(12):2330–2335, 2009. doi:10.1016/j.laa.2009.02.034.

4 Symeon Bozapalidis and Olympia Louscou-Bozapalidou. The rank of a formal tree power series. *Theoretical Computer Science*, 27(1):211–215, 1983. doi:10.1016/0304-3975(83)90100-7.

5 Joel E Cohen and Uriel G Rothblum. Nonnegative ranks, decompositions, and factorizations of nonnegative matrices. *Linear Algebra and its Applications*, 190:149–168, 1993.

6 Ronald de Wolf. Nondeterministic Quantum Query and Communication Complexities. *SIAM J. Comput.*, 32(3):681–699, 2003. doi:10.1137/S0097539702407345.

7 Nathanaël Fijalkow, Guillaume Lagarde, Pierre Ohlmann, and Olivier Serre. Lower bounds for arithmetic circuits via the Hankel matrix. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:180, 2018. URL: https://eccc.weizmann.ac.il/report/2018/180.

8 Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Raj Tiwary, and Ronald de Wolf. Exponential Lower Bounds for Polytopes in Combinatorial Optimization. *J. ACM*, 62(2):17:1–17:23, 2015. doi:10.1145/2716307.

9 Samuel Fiorini, Thomas Rothvoß, and Hans Raj Tiwary. Extended Formulations for Polygons. *Discrete & Computational Geometry*, 48(3):658–668, 2012. doi:10.1007/s00454-012-9421-9.

10 Michel Fliess. Matrices de Hankel. *J. Math. Pures Appl. (9)*, 53:197–222, 1974.

11 Hervé Fournier, Guillaume Malod, Maud Szusterman, and Sébastien Tavenas. Nonnegative rank measures and monotone algebraic branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:100, 2019. URL: https://eccc.weizmann.ac.il/report/2019/100.

12 Peter Frankl and Vojtěch Rödl. Forbidden intersections. *Trans. Amer. Math. Soc.*, 300(1):259–286, 1987. doi:10.2307/2000598.

13 Pavel Hrubes and Amir Yehudayoff. Homogeneous Formulas and Symmetric Polynomials. *Computational Complexity*, 20(3):559–578, 2011. doi:10.1007/s00037-011-0007-3.

14 Pavel Hrubes and Amir Yehudayoff. Formulas are Exponentially Stronger than Monotone Circuits in Non-commutative Setting. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 10–14, 2013. doi:10.1109/CCC.2013.11.

15 Mark Jerrum and Marc Snir. Some Exact Complexity Results for Straight-Line Computations over Semirings. *J. ACM*, 29(3):874–897, July 1982. doi:10.1145/322326.322341.

16 Stasys Jukna and Georg Schnitger. On the optimality of Bellman-Ford-Moore shortest path algorithm. *Theor. Comput. Sci.*, 628:101–109, 2016. doi:10.1016/j.tcs.2016.03.014.

17 Adam Klivans and Amir Shpilka. Learning Restricted Models of Arithmetic Circuits. *Theory of Computing*, 2(10):185–206, 2006. doi:10.4086/toc.2006.v002a010.

18 Mrinal Kumar and Shubhangi Saraf. On the Power of Homogeneous Depth 4 Arithmetic Circuits. *SIAM J. Comput.*, 46(1):336–387, 2017. doi:10.1137/140999335.

19 Noam Nisan. Lower Bounds for Non-Commutative Computation (Extended Abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418, 1991. doi:10.1145/103418.103462.

20 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Comput. Complexity*, 6(3):217–234, 1996/97. doi:10.1007/BF01294256.

21 Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2):8:1–8:17, 2009. doi:10.1145/1502793.1502797.

22 C.P. Schnorr. A lower bound on the number of additions in monotone computations. *Theoretical Computer Science*, 2(3):305–315, 1976. doi:10.1016/0304-3975(76)90083-9.

Unambiguous Catalytic Computation

Chetan Gupta
Indian Institute of Technology Kanpur, India
gchetan@cse.iitk.ac.in

Rahul Jain
Indian Institute of Technology Kanpur, India
jain@cse.iitk.ac.in

Vimal Raj Sharma
Indian Institute of Technology Kanpur, India
vimalraj@cse.iitk.ac.in

Raghunath Tewari
Indian Institute of Technology Kanpur, India
rtewari@cse.iitk.ac.in

Abstract

The catalytic Turing machine is a model of computation defined by Buhrman, Cleve, Koucký, Loff, and Speelman (STOC 2014). Compared to the classical space-bounded Turing machine, this model has an extra space which is filled with arbitrary content in addition to the clean space. In such a model we study if this additional filled space can be used to increase the power of computation or not, with the condition that the initial content of this extra filled space must be restored at the end of the computation.

In this paper, we define the notion of unambiguous catalytic Turing machine and prove that under a standard derandomization assumption, the class of problems solved by an unambiguous catalytic Turing machine is same as the class of problems solved by a general nondeterministic catalytic Turing machine in the logspace setting.

2012 ACM Subject Classification Mathematics of computing

Keywords and phrases Catalytic computation, Logspace, Reinhardt-Allender

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.16

Funding *Rahul Jain*: Ministry of Human Resource Development, Government of India
Raghunath Tewari: DST Inspire Faculty Grant, Visvesvaraya Young Faculty Fellowship

Acknowledgements The fourth author would like to thank Michal Koucký for valuable discussions and for suggesting key ideas which led to the proof of the main result in this paper. The first and third author would like to thank Ministry of Electronics and IT, India for supporting this research through the Visvesvaraya PhD. The authors would also like to thank the anonymous reviewers for their valuable comments which helped in improving the presentation of this paper and suggesting an alternative proof of CNL = coCNL as a corollary of our result.

1 Introduction

The catalytic computational model was first introduced by Buhrman et al. [2]. It is a computational model constructed by equipping a standard Turing machine with a large *auxiliary tape* in addition to its work tape. This auxiliary tape is filled with arbitrary data which must be restored at the end of the computation. A catalytic Turing machine with a workspace of size $s(n)$ can be assumed to have auxiliary space of size $2^{s(n)}$. The question that arises is, whether having access to this additional tape increases the power of computation or not. At first, this extra filled space seemed to be of no use. However, surprisingly, Buhrman et al. [2] showed that there exist some problems which can be solved by a deterministic catalytic logspace Turing machine (CL) but are not known to be solvable by a standard deterministic logspace Turing machine (L), or even nondeterministic logspace NL. More precisely, they showed that $\mathsf{uniformTC}^1 \subseteq \mathsf{CL} \subseteq \mathsf{ZPP}$ and $\mathsf{uniformTC}^1$ is known to contain NL which is believed to be different from L. This result gives the motivation to explore the power of the catalytic Turing machine further.

In a subsequent result, Buhrman et al. [3] defined a nondeterministic catalytic computational model. In a nondeterministic catalytic Turing machine, the content of the auxiliary tape must be restored to its initial content for every sequence of nondeterministic choices. The nondeterministic equivalent of CL is called CNL. Using the same observation as in [2] they showed that $\mathsf{CNL} \subseteq \mathsf{ZPP}$. They also showed that, under a standard derandomization assumption, the class of problems solvable by a nondeterministic logspace catalytic Turing machine (CNL) is closed under complement, that is $\mathsf{CNL} = \mathsf{coCNL}$. To prove this, they first show that on a specific input x most of the configuration graphs of a CNL machine will be of polynomial size. They use the pseudorandom generator of [5] to obtain a polynomial size configuration graph. However, having access to a polynomial size graph is not enough because the size of a vertex in the graph is still exponentially larger than the size of the workspace. To circumvent this problem, they use a *hash function* picked from a hash family which maps these vertices injectively to smaller values. After that, they apply the inductive counting technique of Immerman and Szelepcsényi on this smaller size graph to obtain the final result [4, 8].

In this paper, we define a variant of nondeterministic catalytic Turing machine called *unambiguous* catalytic Turing machine. Analogous to the standard Turing machine, an unambiguous catalytic Turing machine is a nondeterministic catalytic Turing machine which has at most one accepting path on each input.

We show that under the same derandomization assumption as that of [3], in the logspace setting, unambiguous catalytic Turing machine (CUL) and nondeterministic catalytic Turing machine are equivalent in power. This is stated formally in the following theorem.

▶ **Theorem 1** (Main Theorem). *If there exists a constant* $\epsilon > 0$ *such that* $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\epsilon n})$*, then* $\mathsf{CNL} = \mathsf{CUL}$.

We prove Theorem 1 by giving an unambiguous logspace catalytic algorithm which answers if in the configuration graph of a CNL machine, the accepting vertex is reachable from the starting vertex or not. For this, we use (i) the pseudorandom generator used by [1] and [3] to obtain a small size min-unique weighted configuration graph of the CNL machine, (ii) the hashing scheme of [3] which maps the vertices of the configuration graph to smaller values, and (iii) the double counting technique of [7]. Our result is analogous to a result of [1] in the traditional Turing machine model, where authors prove that, if there are problems in $\mathsf{DSPACE}(n)$ which require exponential size circuits, then $\mathsf{UL} = \mathsf{NL}$.

The rest of the paper is organized as follows. Section 2 contains definitions of nondeterministic and unambiguous catalytic computation. We state the derandomization assumption under which our result holds, pseudorandom generators and the hashing scheme that we have used. In Section 3, we prove the main result $\mathsf{CUL} = \mathsf{CNL}$.

2 Preliminaries

In this section, we present the necessary definitions, notations, and lemmas. We start with the definition of a nondeterministic catalytic Turing machine as defined in [3].

▶ **Definition 2.** Let $\mathcal{M}$ be a nondeterministic Turing machine with three tapes: one input tape, one work tape, and one *auxiliary tape*.

Let $x \in \{0,1\}^n$ be an input, and $w \in \{0,1\}^{s_a(n)}$ be the initial content of the auxiliary tape. We say that $\mathcal{M}(x,w)$ accepts x if there exists a sequence of nondeterministic choices that makes the machine accept. If for all possible sequences of nondeterministic choices $\mathcal{M}(x,w)$ does not accept, the machine rejects x.

Then $\mathcal{M}$ is said to be a *catalytic nondeterministic* Turing machine using workspace $s(n)$ and auxiliary space $s_a(n)$ if for all inputs, the following three properties hold.

1. **Space bound**. The machine $\mathcal{M}(x,w)$ uses space $s(n)$ on its work tape and space $s_a(n)$ on its auxiliary tape.
2. **Catalytic condition**. $\mathcal{M}(x,w)$ halts with w on its auxiliary tape, irrespective of its nondeterministic choices.
3. **Consistency**. The outcome of the computation is consistent among all initial aux-tape content w. $\mathcal{M}(x,w)$ should either accept for all choices of w — in which case we say $\mathcal{M}$ accepts x — or it rejects for all possible w — $\mathcal{M}$ rejects x. However, the specific nondeterministic choices that make $\mathcal{M}(x,w)$ go one way or the other may depend on w.

▶ **Definition 3.** $\mathsf{CNSPACE}(s(n))$ is the set of decision problems that can be solved by a nondeterministic catalytic Turing machine with at most $s(n)$ size workspace and $2^{s(n)}$ size auxiliary space. CNL denotes the class $\mathsf{CNSPACE}(O(\log n))$.

Unambiguous computation is a natural restriction of nondeterministic computation where on every input the Turing machine can have at most one nondeterministic path which accepts the input. In the domain of catalytic computation, the definition naturally extends as follows.

▶ **Definition 4.** An *unambiguous* catalytic Turing machine is a nondeterministic catalytic Turing machine which on every input produces at most one sequence of nondeterministic choices where the machine accepts.

▶ **Definition 5.** $\mathsf{CUSPACE}(s(n))$ is the set of decision problems that can be solved by an unambiguous catalytic Turing machine with at most $s(n)$ size workspace and $2^{s(n)}$ size auxiliary space. CUL denotes the class $\mathsf{CUSPACE}(O(\log n))$.

In order to present our result, we will use the notion of *configuration graph.* Configuration graphs of a classical Turing machine are used heavily in proving space-bounded computation results. A modified version of configuration graph was used for catalytic computations by Buhrman et al. in [3]. They defined the configuration graph in the context of catalytic computation in the following way: Let $\mathcal{M}$ be a nondeterministic catalytic Turing machine with $c \log n$ size workspace and n^c size auxiliary space. Then, $\mathcal{G}_{\mathcal{M},x,w}$ denotes the configuration graph of a nondeterministic catalytic Turing machine $\mathcal{M}$ on input x and initial auxiliary content w. Every vertex of $\mathcal{G}_{\mathcal{M},x,w}$ corresponds to a configuration of $\mathcal{M}$ reachable from the initial configuration of $\mathcal{M}$ which consists of the content of the work tape and the auxiliary tape, head positions of all three tapes and the current state. The graph has a directed edge from a vertex ver_1 to a vertex ver_2 if the configuration corresponding to ver_2 can be reached from the configuration corresponding to ver_1 in one step in $\mathcal{M}$. We will denote the number of the vertices in a configuration graph $\mathcal{G}_{\mathcal{M},x,w}$ by $|\mathcal{G}_{\mathcal{M},x,w}|$.

A configuration of a nondeterministic catalytic Turing machine $\mathcal{M}$ with $c \log n$ size workspace and n^c size auxiliary space can be described with at most $c \log n + n^c + \log n + \log(c \log n) + \log n^c + O(1) = O(n^c)$ bits, where we need $c \log n + n^c$ bits for work and auxiliary tape content, $\log n + \log(c \log n) + \log n^c$ bits for the tape heads, and $O(1)$ bits for the state information. Thus, the total number of configurations of $\mathcal{M}$ can be upper bounded by $2^{O(n^c)}$, which also implies $|\mathcal{G}_{\mathcal{M},x,w}| \leq 2^{O(n^c)}$ for an input x and initial auxiliary content w.

In Section 3, we will prove $\mathsf{CUL} = \mathsf{CNL}$ under the same assumption the following derandomization result holds.

▶ **Lemma 6** ([5, 6]). *If there exists a constant $\epsilon > 0$ such that $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\epsilon n})$ then for all constants c there exists a constant c' and a function $G : \{0,1\}^{c' \log n} \to \{0,1\}^n$ computable in $O(\log n)$ space, such that for any circuit C of size n^c*

$$\left| \underset{r \in \{0,1\}^n}{Pr} [C(r) = 1] - \underset{s \in \{0,1\}^{c' \log n}}{Pr} [C(G(s)) = 1] \right| < \frac{1}{n}.$$

Buhrman et al. in [3], showed a way to get a small size configuration graph of a nondeterministic logspace catalytic Turing machine. We will use the following lemma in our result, a stronger version of which was proved in [3].

▶ **Lemma 7** ([3]). *Let $\mathcal{M}$ be a nondeterministic catalytic Turing machine using $c \log n$ size workspace and n^c size auxiliary space. If there exists a constant $\epsilon > 0$ such that $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\epsilon n})$, then there exists a function $G : \{0,1\}^{O(\log n)} \to \{0,1\}^{n^c}$, such that on every input x and initial auxiliary content w, for at least one seed $s \in \{0,1\}^{O(\log n)}$, $|\mathcal{G}_{\mathcal{M},x,w \oplus G(s)}| \leq n^{2c+3}$. Moreover, G is logspace computable. ($w \oplus G(s)$ represents the bitwise XOR of w and $G(s)$)*

Let $\mathcal{G}$ be a directed graph, with vertex set $V(\mathcal{G})$ and edge set $E(\mathcal{G})$. Then, a weight function for $\mathcal{G}$ is a map $W : E(\mathcal{G}) \to \mathbb{N}$ which maps every edge in $E(\mathcal{G})$ to a natural number. Let $\mathcal{G}_W$ denote the weighted graph with respect to the weight function W. We say a weight function is a k-bit weight function if every edge in $E(\mathcal{G})$ gets a weight in the range $[0, 2^k - 1]$. A k-bit weight function for a graph $\mathcal{G}$ of n vertices can be thought of as a kn^2 length binary string $b = b_1 b_2 \dots b_{k.n^2}$. In such a representation, the weight assigned to the ith edge e_i of $\mathcal{G}$ is $W(e_i) = \textsc{Dec}(b_{j+1} b_{j+2} \dots b_{j+k})$, where $j = k.(i-1)$ and $\textsc{Dec}(x)$ is the natural number whose decimal representation is the binary string x.

We say $\mathcal{G}_W$ is *min-unique*, if there is a unique minimum weight path between every pair of vertices in $\mathcal{G}_W$. For any two vertices u and v in $V(\mathcal{G}_W)$, we denote the weight of the minimum weight path from u to v by $\mathsf{dist}(u, v)$. The following lemma implicit in [1] shows that under the assumption of Lemma 6 there exists a logspace computable pseudorandom generator which gives an $O(\log n)$-bit min-unique weight function for any graph of n vertices.

▶ **Lemma 8.** *If there exists a constant $\epsilon > 0$ such that $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\epsilon n})$, then there exists a logspace computable function $W : \{0,1\}^{O(\log n)} \to \{0,1\}^{c' n^2 \log n}$, such that for any directed graph $\mathcal{G}$ of n vertices there exists at least one seed $s' \in \{0,1\}^{O(\log n)}$ for which $\mathcal{G}_{W(s')}$ is min-unique.*

We also borrow the following lemma about the existence of a hash family from [3].

▶ **Lemma 9** ([3]). *For every n, there exists a family of hash functions $\{h_k\}_{k=1}^{n^3}$, with each h_k a function $\{0,1\}^n \to \{0,1\}^{4 \log n}$, such that the following properties hold. First, h_k is computable in space $O(\log n)$ for every k, and second, for every set $S \subset \{0,1\}^n$ with $|S| \leq n$ there is a hash function in the family that is injective on S.*

▶ **Definition 10.** Let $\mathcal{G}$ be a directed graph, $h : V(\mathcal{G}) \to \{1, 2, \dots, n\}$ be a hash function and W be a weight function for a graph of n vertices. Then, the *hashed-weighted* graph denoted by $\mathcal{G}_{h,W}$ is a weighted graph, such that every edge $uv \in E(\mathcal{G}_{h,W})$ has weight $W(uv) = W(h(u)h(v))$.

3 Reinhardt-Allender's Double Counting in the Catalytic Setting

In this section, we will prove Theorem 1 by constructing a CUL machine $\mathcal{M}'$ which accepts the same language as that of a given CNL machine $\mathcal{M}$. The core idea is to use the double counting technique of Reinhardt and Allender [7] on the configuration graph of $\mathcal{M}$. However, there are few hurdles to implement it.

Firstly note that, as shown by Buhrman et al. [3], the configuration graph of a CNL machine $\mathcal{M}$ can be of exponential size. Therefore, it is not possible for $\mathcal{M}'$ to do double counting on this graph in its workspace, which is just logarithmic in size. To handle this problem, we use the pseudorandom generator of Lemma 7 to get a small size configuration graph. But this does not solve the problem completely as the size of a vertex in the configuration graph is still very large. To solve this, we use the family of hash functions described in Lemma 9. One of these functions injectively maps the vertices of the configuration graph to small hashed values. Both the pseudorandom generator of Lemma 7 and hash function family of Lemma 9 were also used in [3] for performing inductive counting. In our result, to do double counting we need to make the configuration graph min-unique, therefore, we also use the pseudorandom generator of Lemma 8.

In $\mathcal{M}'$, we do the double counting on the configuration graph of $\mathcal{M}$ for every possible triplet consisting of seeds of the pseudorandom generators of Lemma 7 and 8 and a hash function from the hash family of Lemma 9. During the double counting we move on to the next triplet if the hash function doesn't map the vertices injectively or the configuration graph is not min-unique, otherwise, after finishing double counting we accept if the accepting node in the configuration graph was encountered at some point during the process.

We detect if the configuration graph is not min-unique the same way Reinhardt and Allender do it in [7]. Detecting if a hash function doesn't map the vertices injectively is tricky. Note that, to check whether two vertices of the configuration graph have been mapped to the same value or not cannot be done directly by storing them in the workspace of $\mathcal{M}'$. This is because the size of those vertices can be large. Therefore, we perform a clever bit by bit comparison of these vertices to check if they have been mapped to the same value or not. We outline this procedure in Algorithm 3.

In the following lemma, we prove the existence of the pseudorandom generators and the family of hash functions in the context of a configuration graph of a CNL machine.

▶ **Lemma 11.** *Let $\mathcal{M}$ be a nondeterministic catalytic Turing machine using $c \log n$ size workspace and n^c size auxiliary space. For an input x and auxiliary content w, let G be the pseudorandom generator as given in Lemma 7 and s be a seed such that, $|\mathcal{G}_{\mathcal{M},x,w\oplus G(s)}| \leq N$, where $N = n^{2c+3}$. Then,*

1. *there exists a family of logspace computable hash functions $H = \{h_k\}_{k=1}^{O(N^3)}$, such that for each k we have $h_k : \{0,1\}^N \to \{0,1\}^{4\log N}$, and at least one $h_k \in H$ injectively maps $V(\mathcal{G}_{\mathcal{M},x,w\oplus G(s)})$ to $\{0,1\}^{4\log N}$.*
2. *if there exists a constant $\epsilon > 0$ such that $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\epsilon n})$, then there exists a logspace computable function $W : \{0,1\}^{O(\log N^4)} \to \{0,1\}^{c' N^8 \log N^4}$, such that for at least one seed $s' \in \{0,1\}^{O(\log N^4)}$, the hashed-weighted graph $\mathcal{G}_{\mathcal{M},x,w\oplus G(s),h_k,W(s')}$ is min-unique, where h_k injectively maps $V(\mathcal{G}_{\mathcal{M},x,w\oplus G(s)})$ to $\{0,1\}^{4\log N}$.*

Proof. We know that the size of a vertex(configuration) in $\mathcal{G}_{\mathcal{M},x,w\oplus G(s)}$ can be upper bounded by $O(n^c)$. For the sake of simplicity, we assume that $O(n^c) \leq N$. Then, 1 follows directly from Lemma 9 if you take the set S(of Lemma 9) as $V(\mathcal{G}_{\mathcal{M},x,w\oplus G(s)})$.

Now, consider the graph $\mathcal{G}_{\mathcal{M},x,w\oplus G(s)}$ where every vertex is hashed by h_k to some value in the range $[0, N^4 - 1]$ injectively. If we treat this hashed graph as a graph of N^4 many vertices, then 2 follows from Lemma 8. ◀

3.1 Proof of Main Theorem

Since $\mathsf{CUL} \subseteq \mathsf{CNL}$ follows by definition, we only need to show that $\mathsf{CNL} \subseteq \mathsf{CUL}$. Let $\mathcal{M}$ be a nondeterministic catalytic Turing machine with $c \log n$ size workspace and n^c size auxiliary space. We will prove $\mathsf{CNL} \subseteq \mathsf{CUL}$ by showing that there exists an unambiguous catalytic Turing machine $\mathcal{M}'$ with $c' \log n$ size workspace and $n^{c'}$ size auxiliary space, where c' is sufficiently larger than c, such that on every input x and auxiliary content w, $\mathcal{M}(x,w)$ accepts if and only if $\mathcal{M}'(x,w)$ accepts. For the sake of simplicity, we assume that $\mathcal{M}$ has a unique configuration when it accepts an input. Let acc_w and $start_w$ denote the accept and start configurations of $\mathcal{M}$ on input x and auxiliary content w respectively.

Let G, H, W and N be as given in Lemma 11. For $s \in \{0,1\}^{O(\log n)}$, $h_k \in H$, and $s' \in \{0,1\}^{O(\log N^4)}$, we say a triplet $\langle s, h_k, s'\rangle$ is a *good* triplet, if (1) h_k injectively maps the vertices of $\mathcal{G}_{\mathcal{M},x,w\oplus G(s)}$ to $\{0,1\}^{4\log N}$ and (2) $\mathcal{G}_{\mathcal{M},x,w\oplus G(s),h_k,W(s')}$ is min-unique. Otherwise, we call it a *bad* triplet. Existence of a good triplet follows directly from Lemma 11.

In our algorithm for $\mathcal{M}'$, we iterate over all possible combinations of s, h_k, and s'. In each iteration we work with the hashed-weighted configuration graph $\mathcal{G}_{\mathcal{M},x,w\oplus G(s),h_k,W(s')}$. For a good triplet $\langle s, h_k, s'\rangle$, our algorithm **Accept**s if there is a path from $start_{w\oplus G(s)}$ to $acc_{w\oplus G(s)}$. Otherwise, for a bad triplet $\langle s, h_k, s'\rangle$ the algorithm moves on to the next triplet.

To perform the double counting technique on $\mathcal{G}_{\mathcal{M},x,w\oplus G(s),h_k,W(s')}$, we need to identify the vertices which are at distance i from $start_{w\oplus G(s)}$. For this, our algorithm uses an unambiguous procedure Reachable. Another unambiguous procedure Badgraph is used to check if h_k maps $V(\mathcal{G}_{\mathcal{M},x,w\oplus G(s),h_k,W(s')})$ to $\{0,1\}^{4\log N}$ injectively or not.

Algorithm 1 outlines the main algorithm of $\mathcal{M}'$, Algorithm 2 and Algorithm 3 outline the procedures Reachable and Badgraph respectively.

3.1.1 Description of the Algorithm 1

Let x be the input and w be the auxiliary content of $\mathcal{M}'$. We iterate over all triplets $\langle s, h_k, s'\rangle$ using the loop of line 2. In line 3, we set w to $w \oplus G(s)$ and weight function wt to $W(s')$. For the sake of simplicity, we assume that the function wt assigns weight one to every edge. If not we can always split an edge with weight l to an l length path, similar to how it is done in Lemma 2.1 of [7].

Note that from now onwards, we will denote the hashed-weighted graph for the fixed triplet $\langle s, h_k, s'\rangle$ by $\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$. We define two sets $C_{=i}$ and $C_{<i}$ for $\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ as follows:

- $C_{=i} = \{ver \in V(\mathcal{G}_{\mathcal{M},x,w,h_k,wt}) \mid \mathsf{dist}(start_w, ver) = i\}$,
- $C_{<i} = \bigcup_{j=0}^{i-1} C_{=j}$.

For applying double counting technique, we use two counters c_i and d_i, where, $c_i = |C_{<i+1}|$ and $d_i = \Sigma_{ver\in C_{<i+1}} \mathsf{dist}(start_w, ver)$. Clearly, $c_0 = 1$ and $d_0 = 0$. From lines 5 to 19, we compute the counters c_i and d_i iteratively from the values of c_{i-1} and d_{i-1}. Since for a good triplet $\langle s, h_k, s'\rangle$, $|\mathcal{G}_{\mathcal{M},x,w,h_k,wt}|$ can not be more than N^4, we compute c_i's and d_i's for $i = 1$ to M, where $M = N^4$. Note that we set $M = N^4$ for a special case where wt assigns weight one to every edge, otherwise, its value can be different and need to be set accordingly.

Since h_k maps the vertices of $\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ on $\{0,1\}^{4\log N}$, we go over all possible hashed values v from 0 to $M-1$ and check if there is a vertex ver in $\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ such that $\mathsf{dist}(start_w, ver) = i$ and $h_k(ver) = v$, using the procedure Reachable. If there exists such a vertex ver than we increment c_i and d_i accordingly in line 10. Moreover, if ver is an accepting node then we store this information in the $final$ variable.

Algorithm 1 Algorithm of $\mathcal{M}'$.

G and W are as described in Lemma 11. S and S' are the set of seeds for G and W respectively and H is the hash function family. $M = N^4$ is the maximum size of the configuration graph for a good triplet $\langle s, h_k, s' \rangle$.

```
1:  procedure UNAMBIGUOUSSIMULATION(Input $x$, Auxiliary Content $w$)
2:      for $\langle s, h_k, s' \rangle \in S \times H \times S'$ do
3:          $w \leftarrow w \oplus G(s)$, $wt \leftarrow W(s')$, $final \leftarrow$ FALSE
4:          $c_0 \leftarrow 1$, $d_0 \leftarrow 0$
5:          for $i = 1$ to $M$ do
6:              $c_i \leftarrow c_{i-1}$, $d_i \leftarrow d_{i-1}$
7:              for $v = 0$ to $M - 1$ do
8:                  $(found, finalreach) \leftarrow$ REACHABLE$(v, i, h_k, wt, c_{i-1}, d_{i-1}, s)$
9:                  if $found =$ TRUE then
10:                     $c_i \leftarrow c_i + 1$, $d_i \leftarrow d_i + i$
11:                     if $finalreach =$ TRUE then
12:                         $final \leftarrow$ TRUE
13:                     end if
14:                 else if $found =$ BAD then
15:                     $w \leftarrow w \oplus G(s)$
16:                     Jump to line 2
17:                 end if
18:             end for
19:         end for
20:         $w \leftarrow w \oplus G(s)$
21:         if $final =$ TRUE then
22:             Accept
23:         end if
24:     end for
25:     Reject
26: end procedure
```

If the triplet $\langle s, h_k, s' \rangle$ is bad, we catch it while computing the values of c_i's and d_i's using REACHABLE and move to the next triplet after restoring the initial auxiliary content w in line 15 by XORing it again with $G(s)$. If it is good, then the loop of line 2 terminates normally. After which we restore w in line 20 and **Accept** if $final$ is TRUE in line 22 or move to the next triplet if $final$ is FALSE.

Finally, if acc_w is not reachable from $start_w$ for any good triplet $\langle s, h_k, s' \rangle$, we **Reject** in line 25.

3.1.2 Description of the Algorithm 2

REACHABLE$(v, i, h_k, wt, c_{i-1}, d_{i-1}, s)$ is called only if the following conditions are satisfied:

- **Condition A**: All the vertices in $C_{<i}$ have a unique minimum weight path from $start_w$.
- **Condition B**: All the vertices in $C_{<i}$ are injectively mapped to the set $\{0,1\}^{4 \log N}$.

We define the following conditions based on which REACHABLE detects a bad triplet:

- **Condition I**: There exists a vertex $ver1$ in $C_{<i}$ and a vertex $ver2$ in $C_{=i}$, such that $h_k(ver1) = h_k(ver2) = v$. ($h_k$ *doesn't map* $V(\mathcal{G}_{\mathcal{M},x,w,h_k,wt})$ *to* $\{0,1\}^{4 \log n}$ *injectively.*)

- **Condition II**: There exist vertices $ver1$ and $ver2$ in $C_{=i}$ such that $h_k(ver1) = h_k(ver2) = v$. ($h_k$ *doesn't map* $V(\mathcal{G}_{\mathcal{M},x,w,h_k,wt})$ *to* $\{0,1\}^{4\log n}$ *injectively.*)
- **Condition III**: There exists a vertex ver in $C_{=i}$ such that $h_k(ver) = v$ and ver has more than one minimum weight paths from $start_w$. ($\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ *is not min-unique.*)

Reachable is a nondeterministic procedure which **Reject**s on all sequences of nondeterministic choices except one, where it returns one of the following pair of values:

- (BAD, FALSE) if at least one of the **Condition I, II,** and **III** is satisfied.
- (TRUE, TRUE) if none of the **Condition I, II,** or **III** are satisfied and there exists a vertex ver in $C_{=i}$, such that $h_k(ver) = v$ and $ver = acc_w$.
- (TRUE, FALSE) if none of the **Condition I, II,** or **III** are satisfied and there exists a vertex ver in $C_{=i}$, such that $h_k(ver) = v$ but $ver \neq acc_w$.
- (FALSE, FALSE) if none of the **Condition I, II,** or **III** are satisfied and there does not exist a vertex ver in $C_{=i}$, such that $h_k(ver) = v$.

Reachable in line 3-31, guesses the vertices of $C_{<i}$ in ascending order of their hashed values from h_k. In every iteration, it first cleans a portion of $\mathcal{M}'$'s workspace say z and selects $l \leq i-1$ nondeterministically. Then using the workspace z and auxiliary content w of $\mathcal{M}'$ it simulates the machine $\mathcal{M}$ on x and w for l steps.

During a simulation, we denote the current configuration of $\mathcal{M}$ by $(z, w, pos, state)$, where z denotes the work tape content, w denotes the auxiliary content, pos denotes the head positions on the different tapes and $state$ denotes the current state.

To ensure the ascending order, we use the variable h which is initially set to -1. After every simulation, we compare the hashed value of the current configuration to h in line 7. If the order is violated, we continue the simulation until $\mathcal{M}$ halts, restore w and **Reject**. If not, we assign $h_k(z, w, pos, state)$ to h and use it in the next iteration.

In line 13, we store the sum of the distance of all c_{i-1} many guessed vertices from $start_w$ in the variable d. Variable $v_{present}$ intends to store the information about the existence of a vertex ver in $C_{<i}$, such that $h_k(ver) = v$. In line 15, we set $v_{present}$ to TRUE if $h = v$.

In line 17-29, if $l = i-1$ then we increment cnt for every neighbour of the current configuration which hashes to v. We also set $finalreach$ to TRUE if a neighbour of the current configuration is an accepting node acc_w. Later, we use variables cnt and $v_{present}$ to decide the returning value of Reachable. In line 30, we continue the simulation until a halting state is reached to restore the auxiliary content.

Outside the loop, in line 32, we compare d with d_{i-1} and **Reject** if $d \neq d_{i-1}$. Since vertices in $C_{<i}$ have unique minimum weight path from $start_w$ and they were all guessed in ascending order of their hashed value, $d = d_{i-1}$ holds only for one sequence of nondeterministic choices. For a more detailed proof of why $d = d_{i-1}$ holds only for one sequence of nondeterministic choices, one can refer to Theorem 2.2 of [7].

$v_{present}$ = FALSE implies that there is no vertex ver in $C_{<i}$, such that $h_k(ver) = v$. In such a case, we return the appropriate value based on the value of cnt. $cnt = 0$ implies that there is no vertex ver in $C_{=i}$, such that $h_k(ver) = v$, therefore, we return (FALSE, FALSE). $cnt = 1$ implies that there is exactly one vertex ver in $C_{=i}$, such that $h_k(ver) = v$, therefore, we return (TRUE, $finalreach$). $cnt > 1$ implies that either the **Condition II** or **III** satisfies, therefore, we return (BAD, FALSE).

$v_{present}$ = TRUE implies that there is a vertex ver in $C_{<i}$, such that $h_k(ver) = v$. Here again, if $cnt = 0$ we return (FALSE, FALSE). But if $cnt > 0$, we need to check if **Condition I** is satisfied i.e. there is a vertex $ver' \neq ver$ for which we incremented cnt in line 22 when it was encountered through a path of weight i. Note that, in Reinhardt-Allender's algorithm we do not need to check this because there we do not work with hashed graphs.

We call the procedure Badgraph to check if the **Condition I** is satisfied or not. If Badgraph returns TRUE i.e. **Condition I** is satisfied, we return (BAD, FALSE). If Badgraph returns FALSE, then that means that all the vertices for which we incremented *cnt* in line 22 were actually the vertex *ver* encountered through a different path of weight i, hence we return (FALSE, FALSE).

3.1.3 Description of the Algorithm 3

Badgraph is also a nondeterministic procedure which **Reject**s on all sequences of nondeterministic choices except one, where it returns TRUE if **Condition I** is satisfied, else it returns FALSE.

Badgraph is called from Reachable if there is a vertex *ver* in $C_{<i}$, such that $h_k(ver) = v$ and $cnt > 0$. Let F denote the set of all the vertices for which *cnt* was incremented in the line 22 of Reachable.

In Badgraph we compare *ver* with every vertex in F one bit at a time because we cannot store all the bits due to the limited workspace of $\mathcal{M}'$. In line 2, we set g to be the index of the vertex in F we intend to compare with *ver*. In line 3, we set t to be the index of the bits that we intend to compare. Since a vertex is basically a configuration of machine $\mathcal{M}$ on input x and auxiliary content w, we keep $T = O(n^c)$.

From line 4 to 40, we compare the two bits by guessing the vertices of $C_{<i}$ in the same manner as we do in Rechable. In line 17, we store the tth bit of *ver* in *bit*1. To get the gth vertex of F we use the variable cnt' which is set to 0 initially in line 4. We increment cnt' by one every time $l = i - 1$ and neighbour of the current configuration hashes to v. Thus, $cnt' = g$ in line 25 implies that we have the gth vertex of F and we store the tth bit of that vertex in *bit*2.

In line 38, we compare both bits *bit*1 and *bit*2 and if they are unequal then that means that there is at least one vertex in F which is different from *ver* but both have the same hash value. That implies that **Condition I** is satisfied and hence we return TRUE.

If we never encounter unequal bits in line 38, then that means that all the vertices in F are actually the vertex *ver*. Therefore, we return FALSE in line 43.

3.1.4 Correctness of Algorithm 1

We divide the proof of correctness of the Algorithm 1 into two cases:

Case 1 - Triplet $\langle s, h_k, s' \rangle$ is good: We first prove that if triplet $\langle s, h_k, s' \rangle$ is good then given the correct values of c_{i-1} and d_{i-1} the ith iteration of the loop of line 5 correctly computes values of c_i and d_i.

First notice that, since triplet $\langle s, h_k, s' \rangle$ is good, Reachable will never return BAD for any of the v chosen in line 7. Now, for every vertex *ver* in $C_{=i}$, a call to Reachable$(h_k(ver), i, h_k, wt, c_{i-1}, d_{i-1}, s)$ will return (TRUE, $finalreach$) after which we update the values c_i and d_i accordingly in line 10. And for any vertex *ver* not in $C_{=i}$, a call to Reachable$(h_k(ver), i, h_k, wt, c_{i-1}, d_{i-1}, s)$ will return (FALSE, FALSE). Thus at the end of the ith iteration we will have the correct values of c_i and d_i.

Since we start with the correct values of c_0 and d_0, we can say that the loop of line 5 terminates normally with the correct values of c_M and d_M. Now, if the vertex acc_w is present in the graph $\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ such that $\mathsf{dist}(start_w, acc_w) = i$, then $final$ is set to TRUE in the ith iteration of the loop of line 5 when Reachable$(h_k(acc_w), i, h_k, wt, c_{i-1}, d_{i-1}, s)$ is called and it returns (TRUE, TRUE). Following which we halt and **Accept** in line 22 after restoring the initial auxiliary content of $\mathcal{M}'$ by XORing it with $G(s)$. □

Algorithm 2 The Reachable procedure.

Reachable($v, i, h_k, wt, c_{i-1}, d_{i-1}, s$) is called only if **Condition A** and **Condition B** are satisfied. The procedure checks if there exists a $ver \in V(\mathcal{G}_{\mathcal{M},x,w,h_k,wt})$ such that $h_k(ver) = v$, $\mathsf{dist}(start_w, ver) = i$ and $ver = acc_w$.

```
procedure Reachable($v, i, h_k, wt, c_{i-1}, d_{i-1}, s$)
    $d \leftarrow 0$, $h \leftarrow -1$, $v_{present} \leftarrow$ FALSE, $cnt \leftarrow 0$, $finalreach \leftarrow$ FALSE
    for $j = 1$ to $c_{i-1}$ do
        Clean the workspace $z$ for simulation of $\mathcal{M}$.
        Nondeterministically guess $l \leq i - 1$.
        Simulate $\mathcal{M}$ on $(x, w)$ using $z$ as workspace for $l$ steps.
        if $h_k(z, w, pos, state) \leq h$ then
            Continue the simulation until a halting state is reached.
            $w \leftarrow w \oplus G(s)$
            Reject
        end if
        $h \leftarrow h_k(z, w, pos, state)$
        $d \leftarrow d + l$
        if $h = v$ then
            $v_{present} \leftarrow$ TRUE
        end if
        if $l = i - 1$ then
            $q$ = Number of configurations reachable from $(z, w, pos, state)$ in one step
            for $r = 1$ to $q$ do
                Simulate one more step.
                if $h_k(z, w, pos, state) = v$ then
                    $cnt \leftarrow cnt + 1$
                    if $(z, w, pos, state) = acc_w$ then
                        $finalreach \leftarrow$ TRUE
                    end if
                end if
                Simulate a step back.
            end for
        end if
        Continue the simulation until a halting state is reached.
    end for
    if $d \neq d_{i-1}$ then
        $w \leftarrow w \oplus G(s)$
        Reject
    end if
    if $v_{present}$ = FALSE then
        if $cnt = 0$ then return (FALSE, FALSE)
        else if $cnt = 1$ then return (TRUE, $finalreach$)
        else if $cnt > 1$ then return (BAD, FALSE)
        end if
    else
        if $cnt = 0$ then return (FALSE, FALSE)
        else if Badgraph($v, i, h_k, wt, c_{i-1}, d_{i-1}, s, cnt$) = TRUE then
            return (BAD, FALSE)
        else return (FALSE, FALSE)
        end if
    end if
end procedure
```

Algorithm 3 The BADGRAPH procedure.

BADGRAPH$(v, i, h_k, wt, c_{i-1}, d_{i-1}, s, cnt)$ is called only if **Condition A** and **Condition B** are satisfied. The procedure checks if h_k maps $V(\mathcal{G}_{\mathcal{M},x,w,h_k,wt})$ to $\{0,1\}^{4\log n}$ injectively or not.

```
procedure BADGRAPH($v, i, h_k, wt, c_{i-1}, d_{i-1}, s, cnt$)
    for $g = 1$ to $cnt$ do
        for $t = 1$ to $T$ do
            $d \leftarrow 0$, $h \leftarrow -1$, $bit1 \leftarrow 0$, $bit2 \leftarrow 0$, $cnt' \leftarrow 0$
            for $j = 1$ to $c_{i-1}$ do
                Clean the workspace $z$ for simulation of $\mathcal{M}$.
                Nondeterministically guess $l \leq i - 1$.
                Simulate $\mathcal{M}$ on $(x, w)$ using $z$ as workspace for $l$ steps.
                if $h_k(z, w, pos, state) \leq h$ then
                    Continue the simulation until a halting state is reached.
                    $w \leftarrow w \oplus G(s)$
                    Reject
                end if
                $h \leftarrow h_k(z, w, pos, state)$
                $d \leftarrow d + l$
                if $h = v$ then
                    Store the $t$th bit of $(z, w, pos, state)$ in $bit1$.
                end if
                if $l = i - 1$ then
                    $q =$ Number of configurations reachable from $(z, w, pos, state)$
                    for $r = 1$ to $q$ do
                        Simulate one more step.
                        if $h_k(z, w, pos, state) = v$ then
                            $cnt' \leftarrow cnt' + 1$
                            if $cnt' = g$ then
                                Store the $t$th bit of $(z, w, pos, state)$ in $bit2$.
                            end if
                        end if
                        Simulate a step back.
                    end for
                end if
                Continue the simulation until a halting state is reached.
            end for
            if $d \neq d_{i-1}$ then
                $w \leftarrow w \oplus G(s)$
                Reject
            end if
            if $bit1 \neq bit2$ then
                return TRUE
            end if
        end for
    end for
    return FALSE
end procedure
```

Case 2 - Triplet $\langle s, h_k, s' \rangle$ is bad: A triplet $\langle s, h_k, s' \rangle$ is bad if

- **Violation I**: h_k does not injectively map the vertices of $\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ to $\{0,1\}^{4\log N}$.
- **Violation II**: $\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ is not min-unique.

We will show that if both violations occur simultaneously then Algorithm 1 moves to the next triplet without finishing all M iterations of the loop of line 5. The other cases where only one violation occurs can be analysed similarly.

Let ver_1 and ver_2 be two vertices of $\mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ such that (1) $\mathsf{dist}(start_w, ver_1) \leq \mathsf{dist}(start_w, ver_2)$, (2) $h_k(ver_1) = h_k(ver_2)$, and (3) there does not exist any other pair of vertices say ver_3 and ver_4 such that $h_k(ver_3) = h_k(ver_4)$ and $\mathsf{dist}(start_w, ver_3) \leq \mathsf{dist}(start_w, ver_4) < \mathsf{dist}(start_w, ver_2)$. ver_1 and ver_2 exist due to **Violation I**.

Let ver be a vertex which has more than one minimum weight paths from $start_w$ such that there is no other vertex ver' with more than one minimum weight paths from $start_w$ and $\mathsf{dist}(start_w, ver') < \mathsf{dist}(start_w, ver)$. ver exists due to **Violation II**.

Let $\mathsf{dist}(start_w, ver_2) = i$ and $\mathsf{dist}(start_w, ver) = j$. First note that $i \leq M$, because if $i > M$ then the first $M+1$ vertices on the shortest path from $start_w$ to ver_2 are all injevtively mapped to $\{0,1\}^{4\log N}$ which is not possible because $M = N^4 = |\{0,1\}^{4\log N}|$.

Let $i \leq j$, then both **Condition A** and **Condition B** are satisfied for $C_{<i}$, therefore, the first $i-1$ iterations of the loop of line 5 will terminate normally with correct values of c_{i-1} and d_{i-1}. But on the ith iteration $\textsc{Reachable}(h_k(ver_2), i, h_k, wt, c_{i-1}, d_{i-1}, s)$ will return (BAD, FALSE) as **Condition I** or **II** are satisfied and Algorithm 1 will move on to the next triplet. The case of $j < i$ is similar. □

Finally, if $acc_w \notin \mathcal{G}_{\mathcal{M},x,w,h_k,wt}$ for any good triplet $\langle s, h_k, s' \rangle$ then the value of $final$ is never set to TRUE, therefore, after going over all triplets we **Reject** in line 25.

3.2 coCUL and an alternative proof of CNL = coCNL

Note that, if in line 22 of Algorithm 1 we **Reject** instead of **Accept** after finding an accepting node in the configuration graph for a good triplet $\langle s, h_k, s' \rangle$ and in line 25 we finally **Accept** instead of **Reject** after not finding the accepting node in any of the configuration graph for a good triplet $\langle s, h_k, s' \rangle$, then $L(\mathcal{M}') = \overline{L(\mathcal{M})}$. This proves that $\mathsf{coCNL} \subseteq \mathsf{CUL}(= \mathsf{CNL})$, which implies that $\mathsf{CUL} = \mathsf{CNL} = \mathsf{coCNL} = \mathsf{coCUL}$.

References

1 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, Matching, and Counting Uniform and Nonuniform Upper Bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, October 1999. doi:10.1006/jcss.1999.1646.

2 Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a Full Memory: Catalytic Space. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 857–866, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591874.

3 Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic Space: Nondeterminism and Hierarchy. *Theory of Computing Systems*, 62(1):116–135, January 2018. doi:10.1007/s00224-017-9784-7.

4 Neil Immerman. Nondeterministic Space is Closed Under Complement. *SIAM Journal on Computing*, 17:935–938, 1988.

5 Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 220–229, New York, NY, USA, 1997. ACM. doi:10.1145/258533.258590.

6 Adam R. Klivans and Dieter van Melkebeek. Graph Nonisomorphism Has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses. *SIAM J. Comput.*, 31(5):1501–1526, May 2002. doi:10.1137/S0097539700389652.

7 Klaus Reinhardt and Eric Allender. Making Nondeterminism Unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, February 2000. doi:10.1137/S0097539798339041.

8 Robert Szelepcsényi. The Method of Forced Enumeration for Nondeterministic Automata. *Acta Informatica*, 26:279–284, 1988.

A Fast Exponential Time Algorithm for Max Hamming Distance X3SAT

Gordon Hoi
School of Computing, National University of Singapore, Singapore 117417, Republic of Singapore
e0013185@u.nus.edu

Sanjay Jain
School of Computing, National University of Singapore, Singapore 117417, Republic of Singapore
sanjay@comp.nus.edu.sg

Frank Stephan
Dept. of Mathematics, National University of Singapore, Singapore 119076, Republic of Singapore
School of Computing, National University of Singapore, Singapore 117417, Republic of Singapore
fstephan@comp.nus.edu.sg

Abstract

X3SAT is the problem of whether one can satisfy a given set of clauses with up to three literals such that in every clause, exactly one literal is true and the others are false. A related question is to determine the maximal Hamming distance between two solutions of the instance. Dahllöf provided an algorithm for Maximum Hamming Distance XSAT, which is more complicated than the same problem for X3SAT, with a runtime of $O(1.8348^n)$; Fu, Zhou and Yin considered Maximum Hamming Distance for X3SAT and found for this problem an algorithm with runtime $O(1.6760^n)$. In this paper, we propose an algorithm in $O(1.3298^n)$ time to solve the Max Hamming Distance X3SAT problem; the algorithm actually counts for each k the number of pairs of solutions which have Hamming Distance k.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Design and analysis of algorithms

Keywords and phrases X3SAT Problem, Maximum Hamming Distance of Solutions, Exponential Time Algorithms, DPLL Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.17

Related Version A full version of the paper is available at [8], `https://arxiv.org/abs/1910.01293`.

Funding *Sanjay Jain*: supported in part by the NUS grant C-252-000-087-001 and by Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019 / R1467-000-234-112.
Frank Stephan: supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019 / R1467-000-234-112.

1 Introduction

Given a Boolean formula ϕ in conjunctive normal form, the satisfiability (SAT) problem seeks to know if there are possible truth assignments to the variables such that ϕ evaluates to the value "True". One naïve way to solve this problem is to brute-force all possible truth assignments and see if there exist any assignment that will evaluate ϕ to "True". Suppose that there are n variables and m clauses, we will take up to $O(mn)$ time to check if every clause is satisfiable. However, since there are 2^n different truth assignments, we will take a total of $O(2^n nm)$ time [4]. Classical algorithms were improving on this by exploiting structural properties of the satisfiability problem and in particular its variants. The basic type algorithms are called DPLL algorithms – by the initials of the authors of the corresponding

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 17; pp. 17:1–17:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

papers [12, 11] – and the main idea is to branch the algorithm over variables where one can, from the formula, in each of the branchings deduce consequences which allow to derive values of some further variables as well, so that the overall amount of the run time can be brought down. For the analysis of the runtime of such algorithms, we also refer to the work of Eppstein [2, 3], Fomin and Kratsch [4] and Kullmann [14].

A variant of SAT is the Exact Satisfiability problem (XSAT), where we require that the satisfying assignment has exactly 1 of the literals to be true in each clause, while the other literals in the same clause are assigned false. If we have at most 3 literals per clause with the aim of only having exactly 1 literal to be true, then the whole problem is known as Exact 3-Satisfiability (X3SAT) and this is the problem which we wish to study. Wahlström [10] provided an X3SAT solver which runs in time $O^*(1.0984^n)$ and subsequently there were only slight improvements; here n is, as also always below, the number of variables of the given instance and $O^*(g(n))$ is the class of all functions f bounded by some polynomial $p(\cdot)$ (in the size of the input) times $g(n)$. The problems mentioned before, SAT, 3SAT and X3SAT are all known to be NP-complete. More background information to the above bounds can be found in the PhD theses and books of Dahllöf [19], Gaspers [5] and Wahlström [10].

The runtimes of the problems SAT, 3SAT, XSAT and X3SAT have been well-explored. Sometimes, instead of just finding a solution instance to a problem, we are interested in finding many "diverse" solutions to a problem instance. Generating "diverse" solutions is of much importance in the real world and can be seen in areas such as Automated Planning, Path Planning and Constraint Programming [17]. How does one then measure the "diversity" of solutions? This combinatorial aspect can be investigated naturally with the notion of the Hamming Distance [16]. Given any two satisfying assignments to a satisfiability problem, the Hamming Distance problem seeks to find the number of variables that differ between them. The Max Hamming Distance problem therefore seeks to compute the maximum number of variables that will defer between any two satisfying assignments. If we are interested in the "diversity" of exact satisfying assignments, then the problem is defined as Max Hamming Distance XSAT (X3SAT) accordingly. The algorithm given in this paper actually provides information about the number of pairs of solutions which have Hamming distance k, for $k = 0, 1, \ldots, n$, which could potentially have uses in other fields such as error correction.

A number of authors have worked in these area previously as well. Crescenzi and Rossi [15] as well as Angelsmark and Thapper [13] studied the question to determine the maximum Hamming distance of solutions of instances of certain problems. Dahllöf [18, 19] gave two algorithms for Max Hamming Distance XSAT problem in $O^*(2^n)$ and an improved version in $O^*(1.8348^n)$. The first algorithm enumerates all possible subset of all sizes while checking that they meet certain conditions. The second algorithm uses techniques found in DPLL algorithms. Fu, Zhou and Yin [9] specialised on the X3SAT problem and provided an algorithm to determine the Max Hamming Distance of two solutions of an X3SAT instance in time $O^*(1.676^n)$. Recently, Hoi and Stephan [7] gave an algorithm to solve the Max Hamming Distance XSAT problem in $O(1.4983^n)$.

The main objective of this paper is to propose an algorithm in $O(1.3298^n)$ time to solve the Max Hamming Distance X3SAT problem. The output of the algorithm is a polynomial p which gives information about the number a_k of pairs of solutions of Hamming distance k, for $k = 0, 1, \ldots, n$. The algorithm does so by simplifying in parallel two versions ϕ_1, ϕ_2 of the input instance and the main novelty of this algorithm is to maintain the same structure of ϕ_1 and ϕ_2 and to also hold information about the Hamming distance of the current and resolved variables while carrying out an DPLL style branching algorithm.

2 Basic Approach

Suppose a X3SAT formula ϕ over the set of n variables X is given. The aim is to find the largest Hamming distance possible between two possible value assignments β_1, β_2 to the variables which are solutions of ϕ, that is, make true exactly one literal in each clause of ϕ.

To this end, the algorithm presented in this paper computes a polynomial (called *HD-polynomial*) in u, with degree at most n, such that the coefficient c_k of u^k gives the number of solution pairs (β_1, β_2) such that the Hamming distance between β_1 and β_2 is k. The degree of this polynomial will then provide the largest Hamming distance between any pair of solutions.

▶ **Example 1.** We consider the formula $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_1 \vee x_6 \vee x_7) \wedge (x_2 \vee x_4 \vee \neg x_6)$. Exhaustive search gives for this X3SAT formula the following four solutions:

x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	0	0	0	0
0	1	0	0	1	1	0
0	0	1	1	0	1	0
0	0	1	0	1	0	1

So there are 16 pairs of solutions among which four pairs have Hamming distance 0 and twelve pairs of Hamming distance 4. The intended output of the algorithm is the polynomial $12u^4 + 4u^0$ which indicates that there are four pairs of Hamming distance 0 and twelve pairs of Hamming distance 4.

The reason for choosing this representation is that our algorithm often needs to add/multiply possible partial solutions, which can be done easily using these polynomials whenever needed.

The brute force approach would be to consider a search tree, with four branches at the internal nodes – $(0,0), (0,1), (1,0), (1,1)$ based on values assigned to some variable x for the two possible solutions being compared. If at a leaf the candidate value assignments (β_1, β_2) formed by using the values chosen along the path from the root are indeed both solutions for ϕ and their Hamming distance is k, then the polynomial calculated at the leaf would be u^k; if any of (β_1, β_2) are not solutions then the polynomial calculated at the leaf would be 0. Then, one adds up all the polynomials at the leaves to get the result. This exhaustive search has time complexity (number of leaves) $\times$ $poly(n, |\phi|) = 4^n \times poly(n, |\phi|)$ for n variables.

For $x \in X$ and $i, j \in \{0,1\}$, let $q_{x,i,j}$ be u if $i \neq j$ and 1 otherwise. The above brute force approach for computing the HD-polynomial would be equivalent to computing

$$\sum_{(\beta_1,\beta_2)} \prod_{x \in X} q_{x,\beta_1(x),\beta_2(x)},$$

where (β_1, β_2) in the summation ranges over the pair of solutions for the X3SAT problem ϕ.

However, we may not always need to do the full search as above. We will be using a DPLL type algorithm, where we use branching as above, and simplifications at various points to reduce the number of leaves in the search tree. Note that the complexity of such algorithms is proportional to the number of leaves, modulo a polynomial factor: that is, complexity is $O(poly(n, |\phi|) \times (\text{number of leaves in the search tree})) = O^*(\text{number of leaves in the search tree})$.

As an illustration we consider some examples where the problems can be simplified. If there is a clause (x, y), then $x = \neg y$ for any solution which satisfies the clause. Thus, x and y's values are linked to each other, and we only need to explore the possibilities for y and can drop the branching for x (in addition one needs to do some book-keeping to make sure the difference in the values of y in two solutions also takes care of the difference in the values of x in the two solutions; this book-keeping will be explained below). As another example, if there is a clause (x, x, z), then value of x must be 0 in any solution which satisfies the clause. Our algorithm would use several such simplifications to bring down the complexity of finding the largest Hamming distance. In the simplification process, we will either fix values of some of the variables, or link some variables as above, or branch on a variable x to restrict possibilities of other variables in clauses involving x and so on (more details below).

In the process, we need to maintain that the HD-polynomial generated is as required. Intuitively, if we consider a polynomial calculated at any node as the sum of the values of the polynomials in the leaves which are its descendant, then the value of the polynomial calculated at the root of the search tree gives the HD-polynomial we want. For this purpose, we will keep track of polynomials named p_{main} and $p_{x,i,j}$, which start with p_{main} being 1, and polynomials $p_{x,i,j} = q_{x,i,j}$, for $x \in X, i, j \in \{0, 1\}$ (here $q_{x,i,j}$ is u for $i \neq j$, and 1 otherwise). If there is no simplification done, then at the leaves, the polynomial p_{main} will become the product of $p_{x,i,j}$, $x \in X$, for the values (i, j) taken by x for the two solutions in that branch. When doing simplification via linking of variables, or assigning truth value to some variables, etc. we will update these polynomials, so as to maintain that the polynomial calculated at the root using above method is the HD-polynomial we need. More details on this updating would be given in the following section.

3 Algorithm for Computing HD-polynomial

In this section we describe the algorithm for finding the HD-polynomial for any X3SAT formula ϕ. Note that we consider clause (x, y, z) to be same as (y, x, z), that is order of the literals in the clause does not matter. We start with some definitions.

Notation: For a formula ϕ with variable x, we use the notation $\phi[x = i]$ to denote the formula obtained by replacing all occurence of x in ϕ by i. Similarly, for a set P containing values/definitions of some parameters, including p_1, p_2, we use $P[p_1 = f, p_2 = g]$ to denote the modification of p_1 to f, p_2 to g (and rest of the parameters remaining the same).

▶ **Definition 2.** *Fix a formula ϕ:*

(a) *For a literal / variable x, x' and x'' and other primed versions are either x or $\neg x$, i.e., they use the same variable x, which may or may not be negated.*

(b) *Two clauses c, c' are called* neighbours *if they share a common variable. For example, (x, y, z) and $(\neg x, w, r)$ are neighbours.*

(c) *Two clauses are called* similar *if one of them can be obtained from the other just by negating some of the literals. They are called* dissimilar *if they are not similar. For example, (x, y) is similar to $(x, \neg y)$, $(1, x, y)$ is similar to $(0, \neg x, y)$, (x, z) is dissimilar to (x, y) and $(x, \neg x, z)$ is dissimilar to $(x, z, \neg z)$.*

(d) *Two X3SAT formulas have the* same structure *if they have the same number of clauses and there is a 1–1 mapping between these clauses such that the mapping maps a clause to a similar clause.*

(e) *A set of clauses C is called* isolated *(in ϕ), if none of the clauses in C is a neighbour of any clause in ϕ which is not in C.*

(f) *A set I of variables is* semisolated *in ϕ* by *J if all the clauses in ϕ either contain only variables from $I \cup J$, or do not contain any variable from I. We will be using such I and J for $|I| \leq 10$ and $|J| \leq 3$ only to simplify some cases.*

(g) *We say that x is* linked *to y, if we can derive that $x = y$ (respectively, $x = \neg y$) in any possible solution using constantly many clauses of the X3SAT formula ϕ as considered in our case analysis (a constant bound of 20 is enough). In this case we say that value i of x is linked to value i of y (value i of x is linked to value $1 - i$ of y respectively).*

▶ **Definition 3** (see Monien and Preis [1]). *Suppose $G = (V, E)$ is a simple undirected graph. A* balanced bisection *is a mapping $\pi : V \to \{0, 1\}$ such that, for $V_i = \{v : \pi(v) = i\}$, $|V_0|$ and $|V_1|$ differ by at most one. Let $cut(\pi) = |\{(v, w) : v \in V_0, w \in V_1\}|$. The bisection width of G is the smallest $cut(\cdot)$ that can be obtained for a balanced bisection.*

Suppose ϕ is the original X3SAT formula given over n variable set X. Our main (recursive) algorithm is MHD$(\phi_1, \phi_2, s_1, s_2, V, P)$, where ϕ_1, ϕ_2 are formulas with the same structure over variable set $V \subseteq X$, s_1, s_2 are some value assignments to variables from X and P is a collection of polynomials (over u) for p_{main} and $p_{x,i,j}$, $x \in X$, $i, j \in \{0, 1\}$. Intuitively, p_{main} represents the portion of the polynomial which is formed using variables which have already been fixed (or implied) based on earlier branching decisions.

Initially, algorithm starts with MHD$(\phi_1 = \phi, \phi_2 = \phi, V = X, s_1 = \emptyset, s_2 = \emptyset, P)$, where ϕ is the original formula given for which we want to find the Hamming distance, X is the set of variables for ϕ, s_1, s_2 are empty value assignments, $p_{main} = 1$, $p_{x,i,j} = q_{x,i,j}$.

Intuitively, the function MHD$(\phi_1, \phi_2, s_1, s_2, V, P)$ returns the polynomial $p_{main} \times \sum_{(\beta_1,\beta_2)} \prod_{x \in V} [p_{x,\beta_1(x),\beta_2(x)}]$, where β_1, β_2 range over value assignments to variables in V which are satisfying for the formula ϕ_1 and ϕ_2 respectively, and which are consistent with the value assignment in s_1, s_2, if any, respectively. Thus, if we consider the search tree, then the node representing MHD$(\phi_1, \phi_2, s_1, s_2, V, P)$ basically represents the polynomial formed

$$\sum_{(\beta_1,\beta_2)} \prod_{x \in X} q_{x,\beta_1(x),\beta_2(x)},$$

where (β_1, β_2) in the summation ranges over the pair of solutions for the X3SAT problem ϕ, consistent with the choices taken for the branching variables in the path from the root to the node. Over the course of the algorithm, the following steps will be done:

(a) using polynomial amount of work (in size of ϕ) branch over some variable or group of variables. That is, if we branch over variable x, we consider all possible values for x in $\{0, 1\}$ for ϕ_1, ϕ_2 (consistent with $s_1(x), s_2(x)$ respectively), and then evaluate the corresponding subproblems: note that MHD$(\phi_1, \phi_2, s_1, s_2, V, P)$ would be the sum of the answers returned by (upto) four subproblems created as above: where in the subproblem for x being fixed to (i, j) in (ϕ_1, ϕ_2) respectively, p_{main} gets multiplied by $p_{x,i,j}$ and x is dropped from V.

(b) simplify the problem, using polynomial (in size of ϕ) amount of work, to MHD$(\phi'_1, \phi'_2, s'_1, s'_2, V', P')$, where we reduce the number of variables in V or the number of clauses in ϕ'_1, ϕ'_2.

Note that all our branching/simplication rules will maintain the correctness of calculation of MHD$(\ldots)$ as described above.

Thus, the overall complexity of the algorithm is $O(poly(n, |\phi|) \times$ [number of leaves in search tree]). In the analysis below thus, whenever branching occurs, reducing the number of variables from n to $n - r_1, n - r_2, \ldots, n - r_k$ in various branches, then we give a corresponding α_0 such that for all $\alpha \geq \alpha_0$, $\alpha^n \geq \alpha^{n-r_1} + \alpha^{n-r_2} + \ldots \alpha^{n-r_k}$. Having these α_0's for each of the cases below would thus give us that the overall complexity of the algorithm is at most $O(poly(n, |\phi|) * \alpha_1^n)$, for any α_1 larger than any of the α_0's used in the cases.

All of our modifications done via case analysis below would convert similar clauses to similar clauses. Thus, if one starts with $\phi_1 = \phi_2$, then as we proceed with the modifications below, the corresponding clauses in the modified ϕ_1, ϕ_2 would remain similar (or both dropped) in the new (sub)problems created. Thus, ϕ_1, ϕ_2 will always have the same structure.

Our algorithm/analysis is based on two main cases. Initially, first case is applied until it can no longer be applied. Then, Case 2 applies, repeatedly to solve the problem (Case 2 will use simplifications as in Case 1.(i) to (iv), but no branching from Case 1). The basic outline of the algorithm is given below, followed by the detailed case analysis.

Algorithm 1 Algorithm Max Hamming Distance X3SAT: MHD($\phi_1, \phi_2, V, s_1, s_2, P$).

```
Output: The polynomial $p_{main} \times \sum_{(\beta_1,\beta_2)} \prod_{x \in V} [p_{x,\beta_1(x),\beta_2(x)}]$, where $\beta_1, \beta_2$ range over
value assignments to variables in $V$ which are satisfying for the formula $\phi_1$ and $\phi_2$
respectively, and which are consistent with the value assignment in $s_1, s_2$, if any, respectively.
Note: As $\phi_1, \phi_2$ have the same structure, the statements below about two clauses being
neighbours, or involving $k$-variables (and other similar questions) have the same answer
for both $\phi_1, \phi_2$.
if (some clause cannot be satisfied (for example $(0,0,0)$ or $(1,x,\neg x)$) in $\phi_1$ or $\phi_2$) then
  return 0. This is Case 1.(i).
else if (for some variable $x \in V$, $s_1(x)$ and $s_2(x)$ are both defined) or ($x$ does not appear
in any of the clauses) then
  return MHD($\phi_1, \phi_2, s_1, s_2, V - \{x\}, P[p_{main} = p_{main} \times (\sum_{i,j} p_{x,i,j})]$), where summation
  is over pairs of $(i,j)$ which are consistent with $(s_1(x), s_2(x))$ (if defined). This is Case
  1.(ii).
else if (some clause contains at most two different variables in its literals) then
  simplify ($\phi_1$, $\phi_2$) according to Case 1.(iii) and return the answer from the updated MHD
  problem.
else if (there are two clauses sharing exactly 2 common variables) then
  simplify ($\phi_1$, $\phi_2$) according to Case 1.(iv) and return the answer from the updated MHD
  problem.
else if (there is a variable appearing in at least 4 dissimilar clauses) then
  branch on this variable and do follow-up linking of the variables according to Case 1.(v),
  return the sum of the answers obtained from the subproblems.
else if (there is a clause with at least four dissimilar neighbours and there is a small set $I$
of variables which are semiisolated by a small set $J$ of variables and conditions prescribed
in Case 1.(vi) below hold; we use this only if $|I| \leq 10, |J| \leq 3$) then
  branch on all variables except one in $J$ and simplify according to Case 1.(vi) and return
  the sum of the answers obtained from the subproblems.
else if (there is a clause with at least 4 dissimilar neighbouring clauses) then
  branch on upto three variables and do follow-up linking according to Case 1.(vii) and
  return the sum of the answers from the subproblems.
else
  In this case all the clauses have at most three dissimilar neighbours, no variable appears
  in more than 3 dissimilar clauses and each clause has exactly three variables and no two
  dissimilar clauses share two or more variables.
  As described in Case 2 below, one can branch on some variables and after simplification,
  have two sets of clauses in $\phi_1$ ($\phi_2$) which have no common variables. Furthermore, as
  the clauses do not satisfy the preconditions for Case 1, they again fall in Case 2, and
  we can repeatedly branch/simplify the formulas until the number of variables/clauses
  become small enough to use brute force.
end if
```

3.1 Case 1

This case applies when either some clause is not satisfiable irrespective of the values of the variables (case (i)) or some variable in V's value has already been determined for both ϕ_1, ϕ_2 (case (ii)) or some clauses in ϕ_1 (and thus ϕ_2) use only one or two variables (case (iii)), or two dissimilar clauses have two common variables (case (iv)), or some variable appears in four dissimilar clauses (case (v)) or some clause has four dissimilar clauses as neighbours (which is divided into two subcases (vi) and (vii) below for ease of analysis).

The subcases here are in order of priority. So (i) has higher priority than (ii) and (ii) has higher priority than (iii) and so on.

(i) If there is a clause which cannot be satisfied (for example the clauses $(0,0,0)$ or $(1,1,x)$ or $(1,x,\neg x)$) whatever the assignment of values to the variables consistent with s_1, s_2 in either ϕ_1 or ϕ_2 respectively, then $\mathrm{MHD}(\phi_1, \phi_2, s_1, s_2, V, P) = 0$.

(ii) If a variable $x \in V$ is determined in both ϕ_1, ϕ_2 (i.e., $s_1(x)$ and $s_2(x)$ are defined), or variable x does not appear in any of the clauses, then do the simplification: update p_{main} to $p_{main} \times (\sum_{i,j} p_{x,i,j})$, where i, j range over value assignments to x in ϕ_1, ϕ_2 which are consistent with $(s_1(x), s_2(x))$ (if defined) respectively. That is, answer returned in this case is $\mathrm{MHD}(\phi_1[x = s_1(x)], \phi_2[x = s_2(x)], s_1, s_2, V - \{x\}, P[p_{main} = p_{main} \times (\sum_{i,j} p_{x,i,j}))$, where the summation is over i, j consistent with $s_1(x), s_2(x)$, if defined.

(iii) If there is a clause which contains only one variable. Then, either the value of the variable is determined (for example when the clause is of the form $(x, \neg x, \neg x)$ or (x), for some literal x, which is satisfiable only via $x = 1$), or the clause is unsatisfiable (for example when it is of the form (x, x) or (x, x, x) – in which case we have that $\mathrm{MHD}(\phi_1, \phi_2, s_1, s_2, V, P) = 0$) or it does not matter what the value of the variable is for the clause to be satisfied (for example, when the clause is $(x, \neg x)$). Thus, we can drop the clause and note down the value of the variable in the corresponding s_i if it is determined (if this is in conflict with the variable having been earlier determined in s_i, then $\mathrm{MHD}(\phi_1, \phi_2, \ldots) = 0$). Note that x may be determined in only one of ϕ_1, ϕ_2, thus we do not update the x appearing in any of the remaining clauses of ϕ_1, ϕ_2 to maintain that the clauses of ϕ_1, ϕ_2 are similar.

If there is a clause which contains literals involving exactly two variables, x and y, then x and y can be linked, either as $x = y$ or $x = \neg y$, as we must have exactly one literal in the clause which is true for any satisfying assignment. Thus, we can replace all usage of y by x (or $\neg x$) in both ϕ_1, ϕ_2, drop the variable y from V and correspondingly, update, for $i, j \in \{0,1\}$, $p_{x,i,j}$ to $p_{x,i,j} \times p_{y,i',j'}$, based on the linking of values i for x in ϕ_1 (j for x in ϕ_2 respectively) to value i' for y in ϕ_1 (j' for y in ϕ_2 respectively). Here, in case value of y is determined in s_1, s_2, then the value of x is correspondingly determined – and in case it is in conflict with an earlier determination then $\mathrm{MHD}(\phi_1, \phi_2, \ldots)$ is 0.

So for below assume no clause has literals involving at most two variables.

(iv) Two clauses share two of the three variables in the literals:

Suppose the clauses in ϕ_1 are (x, y, w) and (x', y', z), where x, x' (similarly, y, y') are literals over same variable.

If $x = x', y = y'$, then we have $w = z$;

If $x = \neg x', y = \neg y'$, then we must have $w = z = 0$;

If $x = x', y = \neg y'$, then we must have $x = 0$ and $w = \neg z$; (case of $x = \neg x'$ and $y = y'$ is symmetrical).

In all the four cases, we have that w is linked to z and thus, z can be replaced using w in both ϕ_1, ϕ_2, with corresponding update of $p_{w,i,j}$ by $p_{w,i,j} \times p_{z,i',j'}$, where i', j' are obtained from i, j based on the linking in ϕ_1, ϕ_2 respectively. Here, in case value of z is determined in s_1, s_2, then the value of w is correspondingly determined – and in case it is in conflict with an earlier determination then $\text{MHD}(\phi_1, \phi_2, \ldots)$ is 0.

(v) A variable x appears in at least four dissimilar clauses.
By Cases 1(iii) and 1(iv), these four clauses use, beside x, variables (y_1, z_1), (y_2, z_2), (y_3, z_3), (y_4, z_4) respectively, which are all different from each other. We branch based on x having values (for (ϕ_1, ϕ_2)): $(0,0), (0,1), (1,0)$ and $(1,1)$. Then, in each of the four clauses involving x, we link the remaining y_i and z_i. Formulas ϕ_1, ϕ_2 and s_1, s_2, V, P are correspondingly updated (that is, x is dropped from V, p_{main} is updated to $p_{main} \times p_{x,i,j}$ based on the branch (i, j), and the linking of the variables is done as in Case 1.(iii)).
Note that for each branch, we thus remove the variable x, and one of the other variables in each of the four clauses. Thus we can remove a total of 5 variables for each subproblem based on the branching for x.

(vi) Though technically we need this case only when some clause has four dissimilar neighbours (see case (vii) and Proposition 4), the simplification can be done in other cases also.
There exists (I, J), $I \cup J \subseteq V$, such that $|I| \leq 10$, $|J| \leq 3$ and (I, J) is semiisolated in ϕ_1 (and thus in ϕ_2 too) and one of the following cases hold.

1. $j = 1$ and $i \geq 1$: Suppose $J = \{x\}$. In this case, we can simplify the formulas ϕ_1, ϕ_2 to remove variables from I as follows:
   Let $W = \{$value vectors (β_1, β_2) with domain $I \cup \{x\}$: β_i is consistent with s_i and all clauses involving variables $I \cup \{x\}$ in ϕ_i are satisfied using $\beta_i\}$.
   Let $W_{i,j} = \{(\beta_1, \beta_2) \in W : \beta_1(x) = i \wedge \beta_2(x) = j\}$.
   Let $p_{x,i,j} = p_{x,i,j} \times (\sum_{(\beta_1,\beta_2) \in W_{i,j}} \prod_{v \in I} p_{v,\beta_1(v),\beta_2(v)})$.
   Let $V = V - I$.
   Remove from ϕ_1 and ϕ_2 all clauses containing variables found in I. If x occurs in any clause after the modification, then answer returned is $\text{MHD}(\phi_1, \phi_2, s_1, s_2, V, P)$, where the parameters are modified as above.
   IF x does not occur in any clause after above modification, then, let $p_{main} = p_{main} \times \sum_{i,j} p_{x,i,j}$, where summation is over values (i, j) for x which are consistent with $(s_1(x), s_2(x))$ if defined. $V = V - I - \{x\}$ and the answer returned is $\text{MHD}(\phi_1, \phi_2, s_1, s_2, V, P)$, where the parameters are modified as above.
   Here note that $j = 0$ case can be similarly handled.
2. $J = \{w, x\}$ and $i \geq 3$, where x appears in some clause C involving a variable not in $I \cup J$.
   In this case, we will branch on x and then using the technique of (vi).1 remove variables from I and then also link the two variables different from x in C. That is, for each $(i, j) \in \{(0,0), (0,1), (1,0), (1,1)\}$, that is consistent with $(s_1(x), s_2(x))$ subproblem $(\phi_{1,i,j}, \phi_{2,i,j}, s_{1,i,j}, s_{2,i,j}, V_{i,j}, P_{i,j})$ is formed as follows:
   (a) Set values of x in ϕ_1 and ϕ_2 as i and j respectively, updating correspondingly p_{main} to $p_{main} \times p_{x,i,j}$ and drop x from the variables V.
   (b) Eliminate I from the subproblem by using the method in (vi).1 (as w is the only element of corresponding J in the subproblem).
   (c) Link the two variables in the clause C which are different from x.
   The answer returned by MHD is the sum of the answers of each of the four subproblems.

Note that in each of the four (or less) subproblems, besides x and members of I, one linked variable in C is removed. Thus, in total at least 5 variables get eliminated in each subproblem.

3. $j = 3$ and $i \geq 4$ and there is a clause which contains at least two variables v, w from J and another variable from I (say the clause is (v', w', e')), where v', w' are literals involving v, w); furthermore v, w appear in clauses involving variables not from I: In this case we will branch on the variables v, w, e (consistent with assignments in s_1, s_2 to these variables if any), and simplify each of the subproblems in a way similar to (vi).2 above. Note that exactly one of (v', w', e') is 1: giving 9 branches based on the three choice for each of ϕ_1 and ϕ_2. The answer returned by MHD is the sum of the answers of each of the (upto) nine subproblems.
Note that apart from the 4 elements of I and v, w, for the clauses using variables not from I, we have two clauses involving v and w. The other variables in each of these clauses can be linked up. Thus, in total for each of the subproblems at least 8 variables are eliminated.

(vii) There exists a clause with at least 4 dissimilar neighbours and none of the above cases apply.
Proposition 4 below argues that there is a clause (x, y, z) (in ϕ_1 and thus in ϕ_2) with at least four neighbours so that further clauses according to one of the following five situations exist (up to renaming of variables):

1. (x', a, b), (x'', c, d), (y', a', c'), $(y'', e, \cdot)$;
2. (x', a, b), (x'', c, d), $(y', e, \cdot)$, $(y'', f, \cdot)$;
3. (x', a, b), (x'', c, d), (y', a', c'), $(z', e, \cdot)$;
4. (x', a, b), (x'', c, d), $(y', e, \cdot)$, $(z', f, \cdot)$;
5. (x', a, b), (x'', c, d), (y', a', e), (z', c', e').

where primed versions of the literals use the same variable as unprimed version (though they maybe negated) and $a, b, c, d, e, f, x, y, z$ are literals involving distinct variables. Here $\cdot$ stand for literals involving variables different from x, y, z, where it does not matter what these variables are, as long as they do not create a situation as in cases 1.(i) to 1.(vi).
Suppose the clause corresponding to (x, y, z) in ϕ_2 is (x''', y''', z'''). Then we branch based on $(x, x''') = (0, 0)$ or $(x, y, z; x''', y''', z''') \in \{ (1,0,0;1,0,0), (1,0,0;0,1,0), (1,0,0;0,0,1), (0,1,0;1,0,0), (0,0,1;1,0,0)\}$. That is either both of x, x''' are 0, or at least one of them is 1 (as before, the branches are only used if the values are consistent with s_1, s_2). The branch based on x being 0 in ϕ_1 and x''' being 0 in ϕ_2 allows us to remove x and three variables from linking y with z, a with b and c with d (a total of four variables). The branch based on the remaining 5 cases allows us to remove x, y, z and four other variables by linking the variables other than x, y, z in each of the neighbouring clause in the five possibilities 1–5 mentioned above (a total of seven variables for each of these subproblems).

▶ **Proposition 4.** *If cases 1.(i) to 1.(vi) above do not apply and if there is a clause with at least four dissimilar neighbours then there is also a clause with neighbours as outlined in (vii).*

Proof. Below primed versions of variables denote a literal involving the same variable – though it may be negated version. Given a clause (x, y, z) with at least four dissimilar neighbours, without loss of generality assume that x, y, z are not negated in this clause (otherwise, we can just interchange them with their negated versions). We let x denote a variable which is in at least two further dissimilar clauses. In the light of Cases 1.(iii), 1.(iv) not applying, these clauses have new variables a, b, c, d, say (x', a, b) and (x'', c, d) (again without loss of generality, a, b, c, d are not negated). In light of Case 1.(v) not applying, variable x is used in no further clause.

If two new variables e, f, different from a, b, c, d, x, y, z appear in some clauses involving x, y, z then there are two clauses of the form (A) $(y', e, \cdot)$ and $(y'/z', f, \cdot)$, or (B) (y', e, f) and $(y'/z', a', c')$ (note that in case (B), both a, b (similarly, both c, d) cannot appear in the clause as case 1.(iv) did not apply). Thus, 1.(vii).2 or 1.(vii).4 (in case (A)) or 1.(vii).1 or 1.(vii).3 (in case (B)) apply.

Now, assume that at most one other variable e, appears in any clause involving x, y, z besides a, b, c, d. Without loss of generality suppose the third neighbour of (x, y, z) was $(y', a', \cdot)$, where $\cdot$ involves variable c or e (it cannot involve b or z as Case 1.(iv) did not apply). Now, if a or b appears in a further outside clause involving a variable other than x, y, z, a, b, c, d, e, then (x', a, b) has neighbours $(x, y, z), (x'', c, d), (a', y', c'/e'), (a''/b', f, \cdot)$ and thus 1.(vii).1, 1.(vii).2, 1.(vii).3 or 1.(vii).4 apply (with interchanging of names of y with a and z with b). If none of a or b appears in a further outside clause involving a variable other than x, y, z, a, b, c, d, e, then one of the cases of 1.(vi) applies with $I \cup J$ being $\{x, y, z, a, b, c, d\}$ or $\{x, y, z, a, b, c, d, e\}$ (based on whether e appears with any of x, y, z or not in some clause), and $J \subseteq \{c, d, e\}$ of the variables which appear in clauses not involving $\{x, y, z, a, b, c, d, e\}$. Here note that in case $J = \{c, d, e\}$, then the side condition of 1.(vi).3 is satisfied using clause (c, d, x''). ◀

3.2 Case 2

This case applies when all clauses have exactly three variables, no two clauses have exactly two variables in common, no variable appears in more than three dissimilar clauses and dissimilar clauses have at most three dissimilar neighbours.

As our operations on similar clauses leaves them similar, for ease of proof writing, we will consider similar clauses in any of the formulas as "one" clause when counting below.

Suppose there are m dissimilar clauses involving n variables. First note that for this case, $m \leq 2n/3$. To see this, suppose we distribute the weight 1 of each variable equally among the dissimilar clauses it belongs to. Then, each clause may get weight $(1/3, 1/2, 1)$ or $(1/2, 1/2, 1/2)$ (or more) based on whether the variables in the clause appear in $(2, 1, 0)$ or $(1, 1, 1)$ other clauses in the worst case. Thus, weight on each clause is at least $3/2$, and thus there are at most $2n/3$ dissimilar clauses.

▶ **Proposition 5.** *For some ϵ_m which goes to 0 as m goes to ∞, the following holds.*

Suppose in ϕ_1 (and thus ϕ_2) there are n variables and m dissimilar clauses each having three literals involving three distinct variables, such that each clause has at most three dissimilar neighbours and each variable appears in at most three dissimilar clauses, and no two dissimilar clauses have two common variables.

Then, we can select $k \leq m(1/6 + \epsilon_m)$ variables, such that branching on all possible values for all of these variables, and then doing simplification based on repeated use of Case 1.(i) to 1.(iv) gives two groups of clauses, each having three literals, where the two groups have no common variables, and

(a) *each clause in each group has at most three dissimilar neighbours,*

(b) *each variable appears in at most three dissimilar clauses,*

(c) *no pair of dissimilar clauses have two common variables,*

(d) *the number of dissimilar clauses in each group is at most $(m - k + 2)/2$.*

Proof. To prove the proposition, consider each dissimilar clause as a vertex, with edge connecting two dissimilar clauses if they have a common variable. Using the bisection width result [5, 6, 1], one can partition the dissimilar clauses into two groups (differing by at most one in cardinality) such that there exist at most $k \leq (1/6 + \epsilon_m) \times m$ edges between the two groups, that is there are at most $(1/6 + \epsilon_m) \times m$ common variables between the two groups of clauses. One can assume without loss of generality that at most one clause has three neighbours on the other side. This holds as if there are two dissimilar clauses, say one in each half, which have all their neighbours on the other side, then we can switch these two clauses to the other side and decrease the size of the cut. On the other hand, if both these clauses (say A and B) belong to the same side, then we can switch A to the other side, and switch the side of one of B's neighbours – this also decreases the size of the cut.

To see that the properties mentioned ((a), (b) and (c)) are preserved, suppose in a clause (x, y, z), we branch on x and thus link y with z; here we assume without loss of generality that x, y, z are all positive literals. Note that as (x, y, z) has at most three neighbours, one of which contains x, there can be at most two other neighbours of the clause (x, y, z) which contain y or z.

First suppose y (respectively z) does not appear in any other clause. Without loss of generality assume that y gets dropped and replaced by z or $\neg z$ based on the linking. Then dropping the clause (x, y, z) and replacing y by z does not increase the number of dissimilar clauses that z appears in, nor does it increase the number of neighbours of these clauses as there is no change in variable name in any clause which is not dropped.

Next suppose both y and z appear in exactly one other dissimilar clause, say (y', a, b) and (z', c, d), where y' and z' are literals involving y and z respectively. In that case, linking y and z (and replacing z by y), makes these two clauses neighbours (if not already so) – which is compensated by the dropping of the neighbour (x, y, z); the number of clauses in which y appears remains two. In case these two clauses were already neighbours (say $a = c$ or $\neg c$), then due to application of Case 1.(iv), b and d get linked, clauses (y, a, b) and (z, c, d) thus become similar (resulting in decrease in the neighbour by one for these clauses) and the above analysis can then be recursively applied for linking b with d.

Now considering the edges (and corresponding common variable for the edge) in the cut, and branching on all these variables (while being consistent with s_1 and s_2) and then doing simplification as in Cases 1(i) to 1(iv), we have that each partition is left with at most $(m+1-(k-1))/2$ dissimilar clauses. This holds as, by our assumption above, except maybe for one clause, all dissimilar clauses have at most two neighbours on the other side. Thus, by linking the remaining variables for each of the clauses involved in the cut, we can remove $(k-1)/2$ dissimilar clauses on each side using Case 1(iii). ◀

Thus, one can recursively apply the above modifications in Case 2 to each of the two groups of clauses, one after other, until all the variables have been assigned the values or linked to other variables (where the leaf cases occur when the number of dissimilar clauses is small enough to use brute force assigning values to all of the variables).

Now we count how many variables need to be branched for Case 2 in total if one starts with m clauses involving n variables. The worst case happens when $k = (1/6+\epsilon_m)m$ and the total number of variables which need to be branched on is $m(1 + 5/12 + 5^2/(12^2) + \ldots) * (1/6 + \epsilon)$, where one can take ϵ as small as desired for corresponding large enough m. Thus the number of variables branching would be $m(2/7 + 12\epsilon/7) \leq n(4/21 + 24\epsilon/21)$. As branching on each variable gives at most 4 children, the number of leaves (and thus complexity of the algorithm based on Case 2) is bounded by $4^{4n/21+o(n)}$.

3.3 Overall Complexity of the Algorithm

Note that modifications in each of the above cases takes polynomial time in the original formula ϕ.

Visualize the running of the above algorithm as a search tree, where the root of the tree is labeled as the starting problem $\text{MHD}(\phi, \phi, V = X, s_1 = \emptyset, s_2 = \emptyset, P)$, with P having $p_{main} = 1$, $p_{x,i,j} = q_{x,i,j}$.

At any node, if a simplification case applies, then the node has only one child with the corresponding updated parameters. If a braching case applies, then the node has children corresponding to the parameters in the branching.

As the work done at each node is polynomial in the length of ϕ, the overall time complexity of the algorithm is $poly(n, |\phi|)\times$ (number of leaves in the above search tree).

We thus analyze the number of possible leaves the search tree would generate.

Suppose $T(r)$ denotes the number of leaves rooted at a node $\text{MHD}(\ldots, V, \ldots)$, where V has r variables.

Case 1.(i) to Case 1.(iv) and Case 1.(vi).1 do not involve any branching.

If Case 1.(v) is applied to a MHD problem involving r variables, then it creates at most four subproblems, each having at most $r-5$ variables. Thus, the number of leaves generated in this case is bounded by $4T(r-5)$. Note that $T(r) = O(\alpha^r)$, for $\alpha \geq \alpha_0 = 1.3196$ satisfies the constraints of this equation.

If Case 1.(vi).2 is applied to a MHD problem involving r variables, then it creates at most 4 subproblems each involving at most $r-5$ variables. Thus, the number of leaves generated in this case is bounded by $4T(r-5)$. Note that $T(r) = O(\alpha^r)$, for $\alpha \geq \alpha_0 = 1.3196$ satisfies the constraints of this equation.

If Case 1.(vi).3 is applied to a MHD problem involving r variables, then it creates at most 9 subproblems each involving at most $r-8$ variables. Thus, the number of leaves generated in this case is bounded by $9T(r-8)$. Note that any $T(r) = O(\alpha^r)$, for $\alpha \geq \alpha_0 = 1.3162$ satisfies the constraints of this equation.

If Case 1.(vii) is applied to a MHD problem involving r variables, then it creates at most 6 subproblems, one involving at most $r-4$ variables and the other involving at most $r-7$ variables. Thus, the number of leaves generated in this case is bounded by $T(r-4)+5T(r-7)$. Note that any $T(r) = O(\alpha^r)$, for $\alpha \geq \alpha_0 = 1.3298$ satisfies the constraints of this equation.

If Case 2 is applied to a MHD problem of r variables, then it creates a search tree which contains at most $O(4^{4r/21+o(r)})$ leaves. Note that any $T(r) = O(\alpha^r)$, for $\alpha \geq \alpha_0 = 1.3023$ satisfies the constraints of this equation.

Thus, the formula $T(r) = O(1.3298^r)$ bounds the number of leaves generated in each of the cases above, for large enough r. Thus, we have the theorem:

▶ **Theorem 6.** *Given a X3SAT formula ϕ, one can find in time $O(poly(n, |\phi|) \times 1.3298^n)$ the maximum hamming distance between any two satisfying assignments for ϕ.*

4 Conclusion and Future Work

In this paper, we considered a branching algorithm to compute the Max Hamming Distance X3SAT in $O(1.3298^n)$ time. Our novelty lies in the preservation of structure at both sides of the formula while we branch.

Our method is faster than the naïve invocation of the Max 2-CSP algorithm (see the discussion in the second-last section of the technical report version of this paper at [8], `https://arxiv.org/abs/1910.01293`). Even if one assumes that every clause has only three neighbours (as in Case 2, but now from the start), the usage of the Max 2-CSP algorithm results in a run-time of $9^{2/15 \times n + o(n)}$ which is contained in $O(1.3404^n)$. Without this assumption, the naïve invocation of the Max 2-CSP algorithm is much worse. Also other invocations of known methods do not give good timebounds.

Our time bound of $O(1.3298^n)$ is achieved by using simple analysis to analyse our branching rules. Our algorithm uses only polynomial space during its computations. This can be seen from the fact that the recursive calls at the branchings are independent and can be sequentialised; each calling instance therefore needs only to store the local data; thus each node of the call tree uses only $h(n)$ space for some polynomial h. The depth of the tree is at most n as each branching reduces the variables by 1; thus the overall space is at most $h(n) \times n$ space.

Furthermore, as we determine the number of pairs of solutions with Hamming distance k for $k = 0, 1, \ldots, n$, where n is the number of variables, one might ask whether this comes with every good algorithm for free or whether there are faster algorithms in the case that one computes merely the maximum Hamming distance of two solutions.

References

1 Burkhard Monien and Robert Preis. Upper bounds on the bisection width of 3- and 4-regular graphs. *Journal of Discrete Algorithms*, 4(3):475–498, 2006.

2 David Eppstein. Small maximal independent sets and faster exact graph coloring. *Proceedings of the Seventh Workshop on Algorithms and Data Structures, Springer Lecture Notes in Computer Science*, 2125:462–470, 2001.

3 David Eppstein. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Transactions on Algorithms*, 2(4):492–509, 2006.

4 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Texts in Theoretical Computer Science, EATCS, Springer, Berlin, Heidelberg, 2010.

5 Serge Gaspers. *Exponential Time Algorithms: Structures, Measures, and Bounds.* VDM Verlag Dr. Müller, 2010.

6 Serge Gaspers and Gregory B. Sorkin. Separate, measure and conquer: faster polynomial-space algorithms for Max 2-CSP and counting dominating sets. *ACM Transactions on Algorithms (TALG)*, 13(4):44:1–44:36, 2017.

7 Gordon Hoi and Frank Stephan. Measure and conquer for Max Hamming Distance XSAT. *International Symposium on Algorithms and Computation, ISAAC 2019, LIPIcs*, 149:18:1–18:20, 2019.

8 Gordon Hoi, Sanjay Jain and Frank Stephan. A fast exponential time algorithm for Max Hamming X3SAT. *arXiv*, 2019. Technical Report version of this paper. `arXiv:1910.01293`.

9 Linlu Fu, Junping Zhou and Minghao Yin. Worst case upper bound for the maximum Hamming distance X3SAT problem. *Journal of Frontiers of Computer Science and Technology*, 6(7):664–671, 2012.

10 Magnus Wahlström. *Algorithms, measures and upper bounds for satisfiability and related problems.* PhD thesis, Department of Computer and Information Science, Linköping University, 2007.

11 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

12 Martin Davis, George Logemann and Donald W. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

13 Ola Angelsmark and Johan Thapper. Algorithms for the maximum Hamming distance problem. *Recent Advances in Constraints, International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 204, Springer Lecture Notes in Computer Science*, 3419:128–141, 2004.

14 Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1–2):1–72, 1999.

15 Pierluigi Crescenzi and Gianluca Rossi. On the Hamming distance of constraint satisfaction problems. *Theoretical Computer Science*, 288(1):85–100, 2002.

16 Richard Wesley Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

17 Satya Gautam Vadlamudi and Subbarao Kambhampati. A combinatorial search perspective on diverse solution generation. *Thirtieth AAAI Conference on Artificial Intelligence*, pages 776–783, 2016.

18 Vilhelm Dahllöf. Algorithms for Max Hamming Exact Satisfiability. *International Symposium on Algorithms and Computation, ISAAC 2005, Springer Lecture Notes in Computer Science*, 3827:829–383, 2005.

19 Vilhelm Dahllöf. *Exact Algorithms for Exact Satisfiability Problems*. PhD thesis, Department of Computer and Information Science, Linköping University, 2006.

Exact and Approximate Digraph Bandwidth

Pallavi Jain
Ben-Gurion University of the Negev, Beer-Sheva, Israel
pallavi@post.bgu.ac.il

Lawqueen Kanesh
The Institute of Mathematical Sciences, HBNI, India
lawqueen@imsc.res.in

William Lochet
University of Bergen, Norway
William.Lochet@uib.no

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, India
saket@imsc.res.in

Roohani Sharma
The Institute of Mathematical Sciences, HBNI, India
saket@imsc.res.in

Abstract

In this paper, we introduce a directed variant of the classical Bandwidth problem and study it from the view-point of moderately exponential time algorithms, both exactly and approximately. Motivated by the definitions of the directed variants of the classical Cutwidth and Pathwidth problems, we define Digraph Bandwidth as follows. Given a digraph D and an ordering σ of its vertices, the *digraph bandwidth* of σ with respect to D is equal to the maximum value of $\sigma(v)-\sigma(u)$ over all arcs (u,v) of D going forward along σ (that is, when $\sigma(u)<\sigma(v)$). The Digraph Bandwidth problem takes as input a digraph D and asks to output an ordering with the minimum digraph bandwidth. The undirected Bandwidth easily reduces to Digraph Bandwidth and thus, it immediately implies that Directed Bandwidth is NP-hard. While an $\mathcal{O}^\star(n!)$[1] time algorithm for the problem is trivial, the goal of this paper is to design algorithms for Digraph Bandwidth which have running times of the form $2^{\mathcal{O}(n)}$. In particular, we obtain the following results. Here, n and m denote the number of vertices and arcs of the input digraph D, respectively.

- Digraph Bandwidth can be solved in $\mathcal{O}^\star(3^n \cdot 2^m)$ time. This result implies a $2^{\mathcal{O}(n)}$ time algorithm on sparse graphs, such as graphs of bounded average degree.
- Let G be the underlying undirected graph of the input digraph. If the treewidth of G is at most t, then Digraph Bandwidth can be solved in time $\mathcal{O}^\star(2^{n+(t+2)\log n})$. This result implies a $2^{n+\mathcal{O}(\sqrt{n}\log n)}$ algorithm for directed planar graphs and, in general, for the class of digraphs whose underlying undirected graph excludes some fixed graph H as a minor.
- Digraph Bandwidth can be solved in $\min\{\mathcal{O}^*(4^n \cdot b^n), \mathcal{O}^*(4^n \cdot 2^{b \log b \log n})\}$ time, where b denotes the optimal digraph bandwidth of D. This allow us to deduce a $2^{\mathcal{O}(n)}$ algorithm in many cases, for example when $b \leq \frac{n}{\log^2 n}$.
- Finally, we give a *(Single) Exponential Time Approximation Scheme* for Digraph Bandwidth. In particular, we show that for any fixed real $\epsilon > 0$, we can find an ordering whose digraph bandwidth is at most $(1+\epsilon)$ times the optimal digraph bandwidth, in time $\mathcal{O}^*(4^n \cdot (\lceil 4/\epsilon \rceil)^n)$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Approximation algorithms

Keywords and phrases directed bandwidth, digraph bandwidth, approximation scheme, exact exponential algorithms

[1] The $\mathcal{O}^\star$ notation hides the polynomial factors in the instance size.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 18; pp. 18:1–18:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.18

Funding *Saket Saurabh*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 819416).

1 Introduction

The Bandwidth problem is a famous combinatorial problem, where given an undirected graph G on n vertices, the goal is to embed its vertices onto an integer line such that the maximum stretch of any edge of G is minimized. More formally, given a graph G on n vertices and an ordering $\sigma : V(G) \rightarrow [n]$, the *bandwidth of σ with respect to G* is $\max_{(u,v)\in E(G)}\{|\sigma(u)-\sigma(v)|\}$. In the Bandwidth problem, given a graph G, the goal is to find an ordering $\sigma : V(G) \rightarrow [n]$, which has *minimum bandwidth* with respect to G. The bandwidth problem has found applications in an array of fields including, but not limited to, the design of faster matrix operations computation on sparse matrices, VLSI circuit design, reducing the search space of constraint satisfaction problems and problems from molecular biology [22]. In many of the real world applications, a fundamental principle that the Bandwidth problem captures is that of delays that occur as a result of allocation of tasks on the time interval that have dependencies among them. An ordering in many scenarios represent the allocation of tasks/objects on a time-line/one-dimensional hardware, and the stretch of an edge captures the delay/effort/expense incurred to reach the other end of the edge.

One restriction on the kind of models captured by Bandwidth is that, the models cannot be tuned to allow for asymmetry or bias. More specifically, what happens when the connections available between the tasks/objects are unidirectional? What happens when there is a bias in terms of delay/expense based on the direction of communication on the time-line/one-dimensional hardware? The above inquisitivities lead to our first contribution to this article, which is the concept of Digraph Bandwidth[2]. Given a directed graph D on n vertices and an ordering $\sigma : V(D) \rightarrow [n]$, the *digraph bandwidth* of σ with respect to D is the *maximum stretch of the forward arcs in the ordering*, that is, $\max_{\substack{(u,v)\in E(D)\\ \sigma(u)<\sigma(v)}}\{\sigma(v)-\sigma(u)\}$. The Digraph Bandwidth problem takes as input a digraph D and outputs an ordering $\sigma : V(D) \rightarrow [n]$ with the least possible *digraph bandwidth* with respect to D.

Observe that, with the introduction of directions in the input graph, Digraph Bandwidth allows us to capture one-way dependencies, that can help in modelling scenarios where the links available for modelling the communication are one-directional. Also, by allowing to care only about the stretch of the forward arcs in the ordering, one can model channels where communication in one direction is cheaper/easier than the other. The later scenarios can occur while modelling an uphill-downhill communication, where the cost of going up is a matter of real concern whereas, the cost of going down is almost negligible.

Note that Digraph Bandwidth is indeed a generalization of the notion of undirected bandwidth, as for any graph G, if $\overleftrightarrow{G}$ denotes the digraph obtained from G by replacing each edge of G by one arc in both direction, then the bandwidth of G is equal to the directed bandwidth of $\overleftrightarrow{G}$. We would like to remark here that on the theoretical front, the way we lift the definition of bandwidth in undirected graphs to directed graphs, by considering the

[2] We choose the name Digraph Bandwidth over the more conventional Directed Bandwidth to avoid clash of names from literature (which will be discussed later).

stretches of *only the forward arcs*, is not something unique that we do for Bandwidth. The idea of only considering arcs going in *one direction* for "optimizing some function" is very common to the directed setting. The simplest such example is the notion of a directed cut. If D is a digraph and X, Y are two disjoint subsets of vertices of D, then the *directed cut* of X and Y, $\texttt{dcut}(X, Y)$, is defined as the set of arcs (u, v) in $E(D)$, where $u \in X$ and $v \in Y$. Another closely related notion is the notion of Directed Cutwidth introduced by Chudnosky et al. [5]. A digraph D on n vertices has cutwidth at most k if there exists an ordering of the vertices σ such that for every $i \in [n-1]$, $\texttt{dcut}(\{\sigma(1), \ldots, \sigma(i)\}, \{\sigma(i+1), \ldots, \sigma(n)\})$ is at most k. Note that our notion of directed bandwidth is a stronger notion than cutwidth, as for any ordering σ, the cutwidth associated to σ is at most the digraph bandwith of σ. There is also a similar notion of Directed Pathwidth [5]. Observe that similar to Directed Cutwidth and Directed Pathwidth, Digraph Bandwidth is 0 on directed acyclic graphs (dags).

We would like to remark that ours is not the first attempt in generalising the definition of bandwidth for digraphs. A notion of bandwidth for directed graphs appeared in 1978 in the paper by Garey et al. [16]. But the notion was defined only for dags. In their problem, which they call Directed Bandwidth (DAG-BW), given a dag D, one is interested in finding a *topological* ordering (a linear ordering of vertices such that for every directed arc (u, v) from vertex u to vertex v, u comes before v in the ordering) of minimum bandwidth. Note that this is very different from our notion of Digraph Bandwidth which is always 0 for dags.

Algorithmic Perspective

Bandwidth is one of the most well-known and extensively studied graph layout problems [17]. The Bandwidth problem is NP-hard [25] and remains NP-hard even on very restricted subclasses of trees, like caterpillars of hair length at most 3 [24]. Furthermore, the bandwidth of a graph is NP-hard to approximate within a constant factor for trees [3]. Polynomial-time algorithms for the exact computation of bandwidth are known for a few graph classes including caterpillars with hair length at most 2 [2], cographs [29], interval graphs [20] and bipartite permutation graphs [19]. A classical algorithm by Saxe [26] solves Bandwidth in time $2^{\mathcal{O}(k)}n^{k+1}$, which is polynomial when k is a constant. In the realm of parameterized complexity, Bandwidth is known to be W[t]-hard for all $t \geq 1$, when parameterized by the bandwidth k of the input graph [4]. However, on trees it admits a parameterized approximation algorithm [12] and an algorithm with running time $2^{\mathcal{O}(k \log k)}n^{\mathcal{O}(1)}$ on AT-free graphs [18]. Unger showed in [27] that the problem is APX-hard. The best known approximation algorithm for this problem is due to Krauthgamer et al. [21] and it provides an $\mathcal{O}(\log^3 n)$ factor approximation.

The Bandwidth problem is one of the test-bed problems in the area of moderately exponential time algorithms and has been studied intensively. Trying all possible permutations of the vertex set yields a simple $\mathcal{O}^*(n!)$ time algorithm while the known algorithms for the problem with running time $2^{\mathcal{O}(n)}$ are far from straightforward. The $\mathcal{O}^*(n!)$ barrier was broken by Feige and Kilian [13] who gave an algorithm with running time $\mathcal{O}^\star(10^n)$. This result was subsequently improved by Cygan and Pilipczuk [6] down to $\mathcal{O}^\star(5^n)$. After a series of improvements, the current fastest known algorithm, due to Cygan and Pilipczuk [9, 7] runs in time $\mathcal{O}^\star(4.383^n)$. We also refer the readers to [8] for the best known exact algorithm running in polynomial space. For graphs of treewidth t, one can design an algorithm with running time $2^n n^{\mathcal{O}(t)}$ [1, 7]. On the other hand, Feige and Talwar [14] showed that the bandwidth of a graph of treewidth at most t can be $(1+\varepsilon)$-approximated in time $2^{\mathcal{O}(\log n(t+\sqrt{\frac{n}{\varepsilon}}))}$. Vassilevska et al. [28] gave a hybrid algorithm which after a polynomial time test, either computes the bandwidth of a graph in time $4^{n+o(n)}$, or provides $\gamma(n)\log^2 n \log\log n$-approximation in polynomial time for any unbounded γ. Moreover, for any two positive integers $k \geq 2$, $r \geq 1$, Cygan and Pilipczuk presented a $(2kr-1)$-approximation algorithm that solves Bandwidth for an arbitrary input graph in $\mathcal{O}(k^{\frac{n}{(k-1)r}} n^{\mathcal{O}(1)})$ time and polynomial space [7]. Finally, Fürer et al. [15] gave a factor 2-approximate algorithm for Bandwidth running in time $\mathcal{O}(1.9797^n)$. DAG-BW, as defined by Garey et al. [16] for dags, was shown to admit a polynomial time algorithm for testing if a dag has bandwidth at most 2. Also, it was proved that the problem to determine if the directed bandwidth of a dag is at most k, for any $k > 2$, is NP-hard even in the case of oriented trees. This notion of directed bandwidth *reappeared* in [23], where it was studied for dense digraphs.

Our Results

The main objective of this paper is to *introduce a directed* variant of the Bandwidth problem for general digraphs and study it from the view point of moderately exponential time algorithms, both *exactly and approximately.* Throughout the remaining, n, m denote the number of vertices and arcs in the input digraph, respectively. For many linear layout problems on graphs on n vertices, beating even the trivial $\mathcal{O}^\star(n!)$ algorithm asymptotically remains a challenge. In this article we design $2^{\mathcal{O}(n)}$ time algorithms for Digraph Bandwidth. Below we mention the challenges that Digraph Bandwidth imposes when we try to apply the techniques used in the design of $2^{\mathcal{O}(n)}$ algorithm for Bandwidth, and how we bend our ways to overcome them to design the desired algorithms.

The $2^{\mathcal{O}(n)}$ time algorithms for Bandwidth that exist in literature (cited above), all follow a common principle of *bucket-then-order.* Suppose one is interested in checking whether the input graph has an ordering of bandwidth b. The *bucket-then-order* procedure is a 2-step procedure, where in the first step, instead of directly guessing the position of the vertex in the ordering, for a range of consecutive positions (called *buckets*) of size $\mathcal{O}(b)$, one guesses the set of vertices that will occupy these positions in the final ordering. This process of allocating a set of vertices to a range of consecutive positions is called *bucketing.* Since one can always assume that the graph is connected, once a bucket for the first vertex is guessed using n trials, its neighbours only have a choice of some c buckets for a small constant c depending on the constant in the order notation of the size of the bucket. This, makes the bucketing step run in time $2^{\mathcal{O}(n)}$. The outcome of the first step is a collection of bucketings which contains a bucketing that is "consistent" with the final ordering. In the second step, given such a consistent bucketing, one can find the final ordering using either a recursive divide and conquer technique or a dynamic programming procedure or a measure and conquer kind of an analysis.

In the case of Digraph Bandwidth, finding a bucketing that is consistent with the final ordering becomes a challenge as even the *information that a vertex is placed in some fixed bucket does not decrease the options of the number of buckets in which its neighbours can be placed.* This is because there could be some out-neighbours (resp. in-neighbours) of it that need to be placed before (resp. after) it thereby contributing to backward arcs, which eventually results in the need for allocating them to far off buckets. We cope up with this challenge of bucketing in two ways - both of which lead to interesting algorithms that run in $2^{\mathcal{O}(n)}$ time in different cases. As a first measure of coping up, we take the strategy of "kill what cause you trouble". Formally speaking, it is the set of backward arcs in the final ordering that have unbounded stretch and hence, make the bucketting process difficult. One way to get back to the easy bucketting case is to guess the set of arcs that will appear as backward arcs in the final ordering. Having guessed these arcs, one can remove them from the graph and preserve the information that the arcs which remain all go forward the final ordering. This problem becomes similar to the DAG-BW problem defined on dags by Garey et al. [16]. We show that one can do the bucketing tricks similar to the undirected case here to design a $2^{\mathcal{O}(n)}$ algorithm for this problem (Theorem 1.1). This together with the initial guessing of the backward arcs gives Theorem 1.2.

▶ **Theorem 1.1.** DAG-BW *on dags can be solved in* $\mathcal{O}^\star(3^n)$ *time.*

▶ **Theorem 1.2.** Digraph Bandwidth *can be solved in* $\mathcal{O}^\star(3^n \cdot 2^m)$ *time.*

Note that even though the 2^m in the running time of Theorem 1.2 looks expensive, it already generates an algorithm better that $\mathcal{O}^\star(n!)$ for any digraph that has at most $o(n \log n)$ arcs. In particular, this implies an exact algorithm with running time $2^{\mathcal{O}(n)}$ whenever $|E(D)| = \mathcal{O}(|V(D)|)$, for example for digraphs with bounded average degree.

We will now briefly explain about our second way of dealing with the bucketing phase. As discussed earlier, getting a hold over the arcs which will go backward in the final ordering, eases out the remaining process. In this strategy, instead of guessing the arcs that goes backward by a brute force way (that takes 2^m), we exploit the fact that guessing a partition of the vertex set into two parts, left and right - which corresponds to the first $n/2$ vertices in the final ordering and the last $n/2$ vertices in the final ordering, also gives hold on *some* if not all backward arcs in the final ordering. We place this simple observation into the framework of a divide and conquer algorithm to get a bucketting that is not necessarily "consistent" with the final ordering, but is not too far away to yield a "close enough" approximation to the optimal ordering. This result is formalized in Theorem 1.3. Effectively, the result states that one can find an ordering whose digraph bandwidth is at most $(1+\epsilon)$ times the optimal in time $\mathcal{O}^\star(1/\epsilon)^n$. Note that, this result is in contrast with the result of Feige and Talwar [14] for undirected bandwidth where they gave an exponential time approximation scheme that run in time which had a dependence on the treewidth of the graph($2^{\mathcal{O}(\log n(t+\sqrt{\frac{n}{\epsilon}}))}$). As a side result of our strategy, we can also design an algorithm for solving Digraph Bandwidth optimally on general digraphs in time $\mathcal{O}^\star(2^{\mathcal{O}(n)} \cdot OPT^n)$ or $\mathcal{O}^\star(2^{\mathcal{O}(n)} \cdot 2^{OPT \log OPT \log n})$, where OPT is the optimal digraph bandwidth of the input digraph. This result is stated in Theorem 1.4. Note that, on one hand where $\mathcal{O}(OPT^n)$ is easy to get for the undirected case (because fixing the position of one vertex in the ordering leaves only $2 \cdot OPT$ choices for its neighbours), it is not trivial for the directed case. Also, observe that Theorem 1.4 gives a $2^{\mathcal{O}(n)}$ algorithm whenever $b \leq {}^{n}\!/\!\log^2 n$.

▶ **Theorem 1.3** ((Single) Exponential Time Approximation Scheme)**.** *For any real number* $\epsilon > 0$*, for any digraph* D*, one can find an ordering of digraph bandwidth at most* $(1+\epsilon)$ *times the optimal, in time* $\mathcal{O}^*(4^n \cdot (\lceil 4/\epsilon \rceil)^n)$*.*

▶ **Theorem 1.4.** Digraph Bandwidth *can be solved in* $\min\{\mathcal{O}^*(4^n \cdot b^n), \mathcal{O}^*(4^n \cdot 2^{b \log b \log n})\}$ *time, where* b *is the optimal digraph bandwidth of the input digraph.*

Our last result is based on the connection of the Bandwidth problem with a subgraph isomorphism problem. Amini et al. [1] viewed the Bandwidth problem, on undirected graphs, as a subgraph isomorphism problem, and using an inclusion-exclusion formula with the techniques of counting homomorphisms on graphs of bounded treewidth, they showed that an optimal bandwidth ordering of a graph on n vertices of treewidth at most t can be computed in time $\mathcal{O}^\star(2^{t \log n + n})$ and space $\mathcal{O}^\star(2^{t \log n})$. Using this approach and by relating Digraph Bandwidth via directed homomorphisms to directed path-like-structures, we obtain the following result.

▶ **Theorem 1.5.** *Let D be a digraph on n vertices and D' be the underlying undirected graph. If the treewidth of D' is at most t, then* Digraph Bandwidth *can be solved in time $\mathcal{O}^\star(2^{n+(t+2)\log n})$.*

Observe that Theorem 1.5 provides $\mathcal{O}^\star(2^{n+\mathcal{O}(\sqrt{n}\log n)})$ algorithm for directed planar graphs and for digraph whose underlying undirected graph excludes some fixed graph H as a minor. This algorithm in fact, yields a $2^{\mathcal{O}(n)}$ time algorithm even when the treewidth of the underlying undirected graph of the given digraph is $\mathcal{O}(n/\log n)$. Notice that Theorem 1.2 gives $2^{\mathcal{O}(n)}$ time algorithm for digraphs of constant average degree, while Theorem 1.5 will not apply to these cases as these digraphs could contain expander graphs of constant degree whose treewidth of the underlying undirected graph could be n/c, for some fixed constant c. On the other hand Theorem 1.5 could give $2^{\mathcal{O}(n)}$ time algorithm for digraphs that have $\mathcal{O}(n^2/\log n)$ arcs but treewidth is $\mathcal{O}(n/\log n)$. Thus, Theorems 1.2 and 1.5 give $2^{\mathcal{O}(n)}$ time algorithm for different families of digraphs.

Road Map

In Section 2, we introduce some notation and definitions. Section 3 is devoted to the proof of Theorem 1.1 and Section 4 proves Theorem 1.2 with the help of Theorem 1.1. Section 5 leads to the proofs of Theorems 1.3 and 1.4. Section 6 proves Theorem 1.5. We finally conclude in Section 7.

2 Preliminaries

For positive integers i, j, $[i] = \{1, \cdots, i\}$ and $[i, j] = \{i, \cdots, j\}$. For any set X, by $X = (X_1, X_2)$ we denote an ordered partition of X, that is $X_1 \cup X_2 = X$, $X_1 \cap X_2 = \emptyset$ and, (X_1, X_2) and (X_2, X_1) are two different partitions of X. For any functions $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$, we say that f_1 *is consistent with* f_2 if for each $x \in X_1 \cap X_2$, $f_1(x) = f_2(x)$. If f_1 and f_2 are consistent, then $f_1 \cup f_2 \colon X_1 \cup X_2 \to Y_1 \cup Y_2$ is defined as $(f_1 \cup f_2)(x) = f_i(x)$, if $x \in X_i$. For any set V of size n, we call a function $\sigma \colon V \to [n]$ as an *ordering* of V). Given an ordering σ of $V(D)$, an arc $(u, v) \in E(D)$ is called a *forward arc in* σ if $\sigma(u) < \sigma(v)$, otherwise it is called a *backward arc*. For a natural number $b \in \mathbb{N}$, we call σ as a *b-ordering of D* if for any forward arc $(u, v) \in E(D)$, $\sigma(v) - \sigma(u) \leq b$, that is, if it has digraph bandwidth at most b. Given a set V and an integer b, a b-bucketing of V is a function $\mathtt{B} : V \to [p, q]$, such that $p, q \in \mathbb{N}$ and for each $i \in [p, q-1]$, $|\mathtt{B}^{-1}(i)| = b$ and $|\mathtt{B}^{-1}(q)| \leq b$. Note that, if $|V|$ is a multiple of b, then $\mathtt{B}^{-1}(q) = b$ and $(q - p + 1) \cdot b = |V|$. If for each $i \in [p, q]$, $|\mathtt{B}^{-1}(i)| \leq b$, we call $\mathtt{B}$ a *partial b-bucketing* of V. Note that, for any b, every b-bucketing is a partial b-bucketing. For a (partial) b-bucketing $\mathtt{B} : V \to [p, q]$, we say that an element $v \in V$ is assigned the *i-th bucket of* $\mathtt{B}$ if $\mathtt{B}(v) = i$ and $\mathtt{B}(v)$ is called the *bucket of* v. Also, b is called the *size of the bucket* $\mathtt{B}(v)$. If $\mathtt{B}(u) = i$ and $\mathtt{B}(v) = j$ and $j > i$, then *the number of buckets between the buckets of* u *and* v is equal to $j - i - 1$. Also, the *number of elements of* V *in the buckets between* i *and* j *is* $(j - i - 1) \cdot b$. In explanations, we sometimes drop b and call $\mathtt{B}$ a *(partial) bucketing* to mean that it is a b-bucketing for some b that should be clear from the context. Given a set V, an integer b and an ordering σ of V, one can associate a b-bucketing with σ which assigns the first b elements in σ the 1-st bucket, the next b-elements the next and so on. This is formalized below. Given a set V, an integer b and an ordering σ of V, we say a b-bucketing $\mathtt{B}$ *respects* σ if $\mathtt{B} : V \to [\lceil |V|/b \rceil]$ is defined as follows. For any $x \in [|V|]$, if $x = ib + j$ for some $i \in \mathbb{N}$ such that $j < b$, then $\mathtt{B}(\sigma_x) = i + 1$ if $j > 0$, and $\mathtt{B}(\sigma_x) = i$ if $j = 0$.

The proofs marked with $\star$ have been omitted because of space constraints and will appear in the full version.

3 Exact Algorithm for Directed Bandwidth for dags

The goal of this section is to prove Theorem 1.1. The algorithm follows the ideas of Cygan and Pilipczuk [10]. We give the details here for the sake of completeness and to mention the little details where we deviate from the algorithm of [10]. Throughout this section, without loss of generality, we can assume that the input digraph D is weakly connected, as otherwise, one can solve the problem on each of the weakly connected components of D and concatenate the orderings obtained from each of them, in any order, to get the final ordering. Also, instead of working on the optimization version of the problem, we work on the decision problem, where together with the input digraph D, one is given an integer b, and the goal is to decide whether there exists a topological ordering of $V(D)$ of bandwidth at most b. It is easy to see that designing an algorithm for this decision version with the desired running time is enough to prove Theorem 1.1. In the following, we abuse notation a little and call (D, b) as an instance of DAG-BW.

Throughout the remaining section, we call a topological ordering of D of bandwidth b as a *b-topological ordering*. A b-bucketing of $V(D)$ is called a *b-topological bucketing* if for all $(u, v) \in E(D)$, either $\mathtt{B}(u) = \mathtt{B}(v)$ or $\mathtt{B}(v) = \mathtt{B}(u)+1$. Our algorithm, like the algorithm of [10], has two phases : Bucketing and Ordering. The Bucketing phase of the algorithm is described by Lemma 3.1.

▶ **Lemma 3.1.** ($\star$) Given an instance (D, b) of DAG-BW, one can find a collection $\mathcal{B}$, of $(b+1)$-topological bucketings of $V(D)$ of size at most $2^{n-1} \cdot \lceil n/b+1 \rceil$, in time $\mathcal{O}^\star(2^n)$, such that for every b-topological ordering σ of D, there exists a bucketing $\mathtt{B} \in \mathcal{B}$ such that $\mathtt{B}$ respects σ.

In the Ordering phase, given a $(b+1)$-topological bucketing $\mathtt{B}$, the algorithm finds a b-topological ordering σ of D, if it exists, such that $\mathtt{B}$ respects σ. From Lemma 3.1, the family $\mathcal{B}$ guarantees the existence of a $(b+1)$-topological bucketing $\mathtt{B}$ of the final desired ordering, if it exists.

To execute this step, we use the idea of finding a sequence of *lexicographically embeddibe sets* using dynamic programming as used in [10]. To define lexicographically embeddible set, the authors first defined the notion of *lexicographic ordering of slots*. We use the same definition in this paper.

▶ **Definition 3.2** (Lexicographic ordering of *slots*). *Given an integer b, let* $\mathtt{bucket}\colon [n] \rightarrow \lceil n/(b+1) \rceil$ *be a function such that* $\mathtt{bucket}(i) = \lceil i/(b+1) \rceil$ *and* $\mathtt{pos}\colon [n] \rightarrow [b+1]$ *be a function such that* $\mathtt{pos}(i) = ((i-1) \mod (b+1)) + 1$. *We define the lexicographic ordering of slots as the lexicographic ordering of* $(\mathtt{pos}(i), \mathtt{bucket}(i))$*, where* $i \in [n]$.

For the Bandwidth problem, the authors of [10] proceed as follows. Given a graph $G = (V, E)$, and a $(b+1)$-bucketing, $\mathtt{B}$ of $V(G)$, they prove that there exists an ordering σ of G such that $\mathtt{B}$ respects σ if and only if there exists a sequence of subsets of $V(G)$, $\emptyset \subset S_1 \subset \cdots \subset S_n$, $|S_i| = i$, for all $i \in [n]$, such that each S_i satisfies the following properties: (i) for each S_i, there is a mapping $\gamma_{S_i} : S_i \rightarrow [n]$ such that $\mathtt{bucket}(\gamma_{S_i}(v)) = \mathtt{B}(v)$, and the set $\{(\mathtt{pos}(\gamma_{S_i}(v)), \mathtt{bucket}(\gamma_{S_i}(v))) \mid v \in S_i\}$ is the set of first $|S_i|$ elements in the lexicographic ordering of slots, and (ii) if $u \in S_i$ and $v \notin S_i$, then $\mathtt{B}(v) \leq \mathtt{B}(u)$. They call such a set S_i as a lexicographically embeddible set. They then obtain $\gamma_{S_{i+1}}$ by extending γ_{S_i} as follows. If $v \in S_i \cap S_{i+1}$, then $\gamma_{S_{i+1}}(v) = \gamma_i(v)$, otherwise $(\mathtt{pos}(\gamma_{S_{i+1}}(v)), \mathtt{bucket}(\gamma_{S_{i+1}}(v)))$ is the $|S_{i+1}|^{\text{th}}$ element in the lexicographic embedding of slots. Recall that there is only one vertex v in $S_{i+1} \setminus S_i$. Furthermore, if v has a neighbor u in S_i, then $\mathtt{B}(v) \leq \mathtt{B}(u)$. If $\mathtt{B}(v) = \mathtt{B}(u)$, then since bucket size is at most $b+1$, $|\gamma(v) - \gamma(u)| \leq b$. If $\mathtt{B}(v) < \mathtt{B}(u)$, then by construction of $\gamma_{S_{i+1}}$, $\texttt{pos}(\gamma_{S_{i+1}}(v)) > \texttt{pos}(\gamma_{S_{i+1}}(u))$. Now, again since each bucket size is at most $b+1$, $|\gamma(v) - \gamma(u)| \leq b$. Therefore, γ_{S_i} can be extended to $\gamma_{S_{i+1}}$. Thus, γ_{S_n} will yield the final ordering. Hence, the goal reduces to finding a sequence $S_1 \subset \cdots \subset S_n$, $|S_i| = i$, for all $i \in [n]$, such that each S_i is a lexicographically embeddible set. We will call such a sequence as a *lexicographically embeddible sequence* (les, in short).

We proceed in a similar way for Directed Bandwidth. We first note that one cannot use the same definition of lexicographically embeddible set as defined above due to the following reason. Suppose that S_i is a lexicographical embeddible set. Consider a vertex $u \in S_i$. Suppose that there exists a vertex $v \notin S_i$ that is an in-neighbor of u, then v cannot belong to the bucket of u as it will not lead to the topological ordering using the method defined above. Also, if v is an out-neighbor of u, then since we are working with topological bucketing, v does not belong to the bucket that precedes the bucket of u. Hence, v belongs to the bucket of u. We also want γ as a topological ordering. Therefore, we redefine the notion of lexicographically embeddible set for Directed Bandwidth as follows.

▶ **Definition 3.3** (Lexicographically embeddible set for digraphs). *Given a $(b+1)$-topological bucketing $\texttt{B}$, of $V(D)$, we say that $S \subseteq V(D)$ is a lexicographically embedibble set if the following condition holds.*

(C1) *For each arc $(u, v) \in E(D)$ such that $u \in S$ and $v \notin S$, $\texttt{B}(u) = \texttt{B}(v)$.*

(C2) *For each arc $(u, v) \in E(D)$ such that $v \in S$ and $u \notin S$, $\texttt{B}(u) = \texttt{B}(v) - 1$.*

(C3) *There exists a b-topological ordering $\gamma\colon S \to [n]$ such that for all $v \in S$, $\texttt{bucket}(\gamma(v)) = \texttt{B}(v)$, and $(\texttt{pos}(\gamma(v)), \texttt{bucket}(\gamma(v)))$ belongs to the first $|S|$ elements in the lexicographic ordering of slots. We refer γ as a partial b-topological ordering that respects lexicographic ordering of slots.*

Now we prove that given a $(b+1)$-topological bucketing $\texttt{B}$ of $V(D)$, to find a b-topological ordering σ such that $\texttt{B}$ respects σ,

▶ **Lemma 3.4.** ($\star$) Given a $(b+1)$-topological bucketing $\texttt{B}$ of $V(D)$, the following are equivalent. **(i)** There exists a b-topological ordering σ of the digraph D such that the unique $(b+1)$-bucketing induced by σ, that is $\texttt{B}_\sigma$, is $\texttt{B}$. In other words, $\texttt{B}_\sigma(v) = \texttt{B}(v)$, for all $v \in V(D)$, **(ii)**There exists a les, $\emptyset \subset S_1 \subset \cdots \subset S_n = V$.

Due to Lemma 3.4, our goal is reduced to find a les. Cygan and Pilipczuk [10] find les using dynamic programming over subsets of the vertex set of given graph. We use a similar dynamic programming approach with appropriate modification because of the revised definition of a lexicographically embeddible set. In the dynamic programming table, for each $S \subseteq V(D)$, $c[S] = 1$, if and only if S is a lexicographically embeddible set. To compute the value of $c[S]$, we first find a vertex $v \in S$ such that $S \setminus \{v\}$ is a lexicographically embeddible set, that is, $c[S \setminus \{v\}] = 1$, and v satisfies the following properties : (i) for all the arcs $(v, u) \in E(D)$ such that $u \notin S$, $\texttt{B}(v) = \texttt{B}(u)$; (ii) for all the arcs $(u, v) \in E(D)$ such that $u \notin S$, $\texttt{B}(u) = \texttt{B}(v) - 1$; and (iii) $\texttt{B}(v) = ((|S| - 1) \mod \lceil n/(b+1) \rceil) + 1$. We compute the value of $c[S]$ for every subset $S \subseteq V(D)$. Note that if $c[V(D)] = 1$, then $V(D)$ is a lexicographically embeddible set. Also, we can compute les by backtracking in the dynamic programming table.

▶ **Lemma 3.5.** ($\star$) Given an instance (D, b) of DAG-BW, and a $(b+1)$-topological bucketing $\texttt{B}$ of $V(D)$ that respects some b-topological ordering of D (if it exists), one can compute a les in time $\mathcal{O}^\star(2^n)$.

Note that using Lemmas 3.1, 3.4 and 3.5, one can solve DAG-BW in $\mathcal{O}^\star(4^n)$ time. The desired running time of $\mathcal{O}^\star(3^n)$ in Theorem 1.1 can be proved by careful analysis of two steps in the algorithm as done in Theorem 12 of [10]. Since the proof of this is the same as that of Theorem 12 of [10], we defer the proof here.

4 Exact Algorithm for Digraph Bandwidth via Directed Bandwidth

We call an ordering of the vertex set of a digraph a b-*ordering* if its digraph bandwidth is at most b. In order to prove Theorem 1.2 observe the following. Let (D, b) be an instance of (the decision version of) DIGRAPH BANDWIDTH. If it is a YES instance, then let σ be a b-ordering of D. Let R be the set of backward arcs in σ. Note that σ is a topological ordering of $D - R$. If we guess the set of backward arcs R in a b-ordering of D (which takes time 2^m), then the goal is reduced to finding a b-topological ordering, σ, of $D - R$ such that if $(u, v) \in R$, then $\sigma(u) > \sigma(v)$. In fact, one can also observe that it is sufficient to find a b-topological ordering, ρ, of $D - R$ such that for all $(u, v) \in R$ either $\rho(u) > \rho(v)$ or $\rho(v) - \rho(u) \leq b$. We claim that we can find the required ordering of $D - R$ using the algorithm for DIRECTED BANDWIDTH for dags given in Section 3. Suppose that σ is a b-ordering of D. Let $\mathtt{B}_\sigma$ be a $(b+1)$-bucketing that respects σ. Let R be the set of backward arcs in σ. Since σ is a b-topological ordering of $D - R$, using Lemma 3.1, $\mathtt{B}_\sigma$ belongs to the collection of $(b + 1)$-bucketings $\mathcal{B}$. Now, using Lemmas 3.5 and 3.4, we obtain a b-topological ordering ρ of $D - R$ that respects $\mathtt{B}_\sigma$. Note that ρ is a b-ordering of D, as for each arc $(u, v) \notin R$, $\rho(v) - \rho(u) \leq b$ because ρ is a b-topological ordering of $D - R$. Also, if $(u, v) \in R$, then observe that if $\mathtt{B}_\sigma(u) \neq \mathtt{B}_\sigma(v)$, then since both σ and ρ respect $\mathtt{B}_\sigma$ and (u, v) is a backward arc in σ, thus, (u, v) is a backward arc in ρ too, that is, $\rho(u) > \rho(v)$. Otherwise, if $(u, v) \notin R$ and $\mathtt{B}_\sigma(u) = \mathtt{B}_\sigma(v)$, then since ρ respects $\mathtt{B}_\sigma$ and the size of the buckets of $\mathtt{B}_\sigma$ is $(b + 1)$, therefore, $|\rho(u) - \rho(v)| \leq b$. Thus, the algorithm of Theorem 1.2 runs the algorithm for DAG-BW for each $R \subseteq E(D)$, to obtain the desired running time.

5 (Single) Exponential Time Approximation Scheme for Digraph Bandwidth

The goal of this section is to prove Theorems 1.3 and 1.4. Let (D, b) be an instance of (the decision version of) DIGRAPH BANDWIDTH. The algorithm relies on an interesting property of a b-bucketing that respects a b-ordering of D. Let σ be a b-ordering of D and let $\mathtt{B}$ be a b-bucketing of $V(D)$ that respects σ. An interesting property of such a bucketing $\mathtt{B}$ is that *if* $(u, v) \in E(D)$, *then either* $\mathtt{B}(u) > \mathtt{B}(v)$ *or* $\mathtt{B}(v) \leq \mathtt{B}(u) + 1$. This is because the size of each bucket in $\mathtt{B}$ is b and σ is a b-ordering of D. Let us call this property of a b-bucketing *useful*. What we saw in the previous section is that if we somehow have a bucketing that respects σ, then one can design an algorithm to fetch σ from this bucketing. In this section, instead of seeking for a bucketing that respects σ we seek for a bucketing with the above mentioned useful property. Observe that, while the existence of such a bucketing with this useful property might not necessarily imply the existence of some b-ordering of D, but having such a bucketing with, for example buckets of size b, definitely yields a $2b$-ordering of D. This is because, given such a bucketing one can assign positions to vertices in the ordering by choosing any arbitrary ordering amongst the vertices that belong to the same bucket and concatenating these orderings in the order of the bucket numbers. By changing the bucket size in the described bucketing, one can yield an ordering of digraph bandwidth at most $(1 + \epsilon)$ times the optimal. This procedure, as we will see, also give an optimal digraph bandwidth ordering when we use the bucket sizes to be 1. Below we give the description of the algorithm used to find a bucketing with the useful property.

We begin by formulating the useful property of a bucketing described above. Since the size of buckets is uniform in a bucketing, instead of defining the property in terms of bucket numbers, we describe it in terms of the number of vertices that can appear between the two buckets corresponding to the end points of a forward arc in the ordering. Such a shift in definition helps us to get slightly better bounds in our running time. For any positive integers b, s and a digraph D, given $X \subseteq V(D)$ and a (partial) b-bucketing of X, say $\mathtt{B} : X \to [p, q]$ for some $p, q \in \mathbb{N}$, we say that the *external stretch of* $\mathtt{B}$ *is at most* s if for each arc $(u, v) \in E(D[X])$, either $\mathtt{B}(u) \geq \mathtt{B}(v)$, or $(\mathtt{B}(v) - \mathtt{B}(u) - 1) \cdot b \leq s$. Recall that $\mathtt{B}(u) - \mathtt{B}(v) - 1$ denote the number of buckets between the bucket of u and the bucket of v. Our major goal now is to prove Lemma 5.1.

▶ **Lemma 5.1.** *Given a digraph D and positive integers b, s, there is an algorithm, that runs in time* $\min\{\mathcal{O}^*(4^n \cdot (\lceil s+1/b \rceil)^n), \mathcal{O}^*(4^n \cdot (\lceil s+1/b \rceil)^{2(b+s)\log n})\}$*, and computes a b-bucketing of $V(D)$, $\mathtt{B} : V(D) \to [\lceil |V(D)|/b \rceil]$, of external stretch at most s.*

We give a recursive algorithm for Lemma 5.1 (Algorithm 1). Since Algorithm 1 is recursive, the input of the algorithm will contain a few more things in addition to D, b, s to maintain the invariants at the recursive steps. We give the description of the input to Algorithm 1 in Definition 5.2.

▶ **Definition 5.2** (Legitimate input for Algorithm 1). *The input $(D, b, s, first, last, left\text{-}bor(V(D)), right\text{-}bor(V(D)), \mathtt{B}_{in})$ is called* legitimate *for Algorithm 1 if the following holds. Let $\delta = \lceil s+1/b \rceil$.*

(P1) *D is a digraph, b, s are positive integers and $|V(D)| = 2^\eta \cdot b \cdot \delta$, where $\eta \geq 0$ is a positive integer.*

(P2) *$first$ and $last$ are positive integers such that $last - first + 1 = 2^\eta$, where η is such that $|V| = 2^\eta \cdot b \cdot \delta$.*

(P3) *$left\text{-}bor(V(D)), right\text{-}bor(V(D)) \subseteq V(D)$,*

(P4) *$\mathtt{B}_{in} : left\text{-}bor(V(D)) \cup right\text{-}bor(V(D)) \to [first, last]$ is a partial b-bucketing such that for each $v \in left\text{-}bor(V(D))$, $\mathtt{B}_{in}(v) \in [first, first + \delta - 1]$, for each $v \in right\text{-}bor(V(D))$, $\mathtt{B}_{in}(v) \in [last - \delta + 1, last]$ and the external stretch of $\mathtt{B}_{in}$ is at most s.*

Observe that $\delta - 1$ represents the number of buckets that can appear between the buckets of u and v in any b-bucketing of external stretch at most s, where the bucket of u precedes the bucket of v and $(u, v) \in E(D)$. We would like to remark that the condition of 1 is not serious as we could have worked without it. We state it like the way we do for the sake of notational and argumentative convenience in the proofs. All it states is that the number of vertices is a power of 2 multiplied by b and δ. The $first$ and $last$ in 2 represents the bucket number of the first and last buckets in the bucketing to be outputted. The relation between $first$ and $last$ in 2 is there to ensure that there are enough buckets to hold the vertices of D. At any recursive call, the sets $left\text{-}bor(V(D))$ and $right\text{-}bor(V(D))$ represent the sets of vertices whose buckets have already been fixed in the previous recursive calls. The set $left\text{-}bor(V)$ represents the set of vertices in V that have an in-neighbour to the vertices that have been decided to be placed in the buckets before the bucket numbered $first$ in the earlier recursive calls. Similarly, $right\text{-}bor(V)$ represents the set of vertices in V that have an out-neighbour to the vertices that have been decided to be placed in the buckets after the bucket numbered $last$ in the earlier recursive calls. Thus, in order to give the final bucketing of external stretch at most s, it is necessary that $left\text{-}bor(V)$ are placed in the first δ buckets and $right\text{-}bor(V)$ are placed in the last δ buckets. This is captured in 4. The next definition describes the properties of the bucketing that would be outputted by Algorithm 1.

▶ **Definition 5.3** (Look-out bucketing for a legitimate instance). *Given a legitimate instance $\mathcal{I} = (D, b, s, first, last, left\text{-}bor(V(D)), right\text{-}bor(V(D)), \mathtt{B}_{in})$, we say a bucketing $\mathtt{B}$ is a* look-out bucketing for *$\mathcal{I}$, if $\mathtt{B}$ is a b-bucketing $\mathtt{B}_{out} : V \to [first, last]$ of external stretch at most s that is consistent with $\mathtt{B}_{in}$.*

Observe that, for the algorithm of Lemma 5.1, a call to Algorithm 1 on $(D, b, s, 1, {}^{|V|}/_{b}, \emptyset, \emptyset, \phi)$ is enough. To give the formal description of Algorithm 1, we will use the following definition.

▶ **Definition 5.4** ($\mathtt{B}$ validates a partition (X_1, X_2))**.** *For any integers p, q and $X' \subseteq X$, let $\mathtt{B} : X' \to [p, q]$ be a partial bucketing of X'. Let (X_1, X_2) be some partition of X. We say that $\mathtt{B}$* validates (X_1, X_2) *if the following holds. Let $r = \lfloor {}^{(p+q)}/_{2} \rfloor$. For each $v \in X_1 \cap X'$, $\mathtt{B}(v) \in [p, r]$ and for each $v \in X_2 \cap X'$, $\mathtt{B}(v) \in [r + 1, q]$.*

Algorithm 1 Algorithm for computing b-bucketing of external stretch at most s.

Input: $\mathcal{I} = (D, b, s, \mathit{first}, \mathit{last}, \mathit{left\text{-}bor}(V), \mathit{right\text{-}bor}(V), B_{in})$ such that $\mathcal{I}$ is legitimate for Algorithm 1.

Output: A look-out bucketing for $\mathcal{I}$, if it exists.

```
Let $V = V(D)$ and $\delta = \lceil {}^{s+1}/_{b} \rceil$.
if $|V| = b \cdot \delta$ then
    return any $b$-bucketing $\mathtt{B} : V \to [\mathit{first}, \mathit{last}]$ that it consistent with $\mathtt{B}_{in}$
Let $\mathit{mid} = {}^{(\mathit{first}+\mathit{last}-1)}/_{2}$.
for each partition $(L, R)$ of $V$ such that $|L| = |R|$ and $B_{in}$ validates $(L, R)$ do
    Let $\mathit{bor}_L = \{v \in L \mid$ there exists $u \in R, (v, u) \in E(D)\}$.
    Let $\mathit{bor}_R = \{v \in R \mid$ there exists $u \in L, (u, v) \in E(D)\}$.
    Let $\mathcal{B}$ be the collection of partial $b$-bucketings, $\mathtt{B} : \mathit{bor}_L \cup \mathit{bor}_R \to [\mathit{mid} - \delta + 1, \mathit{mid} + \delta]$, such that for each $v \in \mathit{bor}_L$, $\mathtt{B}(v) \in [\mathit{mid} - \delta + 1, \mathit{mid}]$, for each $v \in \mathit{bor}_R$, $\mathtt{B}(v) \in [\mathit{mid} + 1, \mathit{mid} + \delta]$, external stretch of $\mathtt{B}$ is at most $s$ and $\mathtt{B}$ is consistent with $\mathtt{B}_{in}$.
    for each $\mathtt{B} \in \mathcal{B}$ do
        Define $\mathtt{B}_{in}^{new} : \mathit{left\text{-}bor}(V) \cup \mathit{bor}_L \cup \mathit{bor}_R \cup \mathit{right\text{-}bor}(V) \to [\mathit{first}, \mathit{last}]$, such that for each $v \in \mathit{left\text{-}bor}(V) \cup \mathit{right\text{-}bor}(V)$, $\mathtt{B}_{in}^{new}(v) = \mathtt{B}_{in}(v)$ and, for each $v \in \mathit{bor}_L \cup \mathit{bor}_R$, $\mathtt{B}_{in}^{new}(v) = \mathtt{B}(v)$.
        Let $\mathtt{B}_{in}^{newL} : \mathit{left\text{-}bor}(V) \cup \mathit{bor}_L \to [\mathit{first}, \mathit{mid}]$ be such that $\mathtt{B}_{in}^{newL} = \mathtt{B}_{in}^{new}{}_{|L}$.
        Let $\mathtt{B}_{in}^{newR} : \mathit{bor}_R \cup \mathit{right\text{-}bor}(V) \to [\mathit{mid} + 1, \mathit{last}]$ be such that $\mathtt{B}_{in}^{newR} = \mathtt{B}_{in}^{new}{}_{|R}$.
        Define $\mathit{left\text{-}bor}(L) = \mathit{left\text{-}bor}(V)$ and $\mathit{right\text{-}bor}(L) = \mathit{bor}_L$.
        Define $\mathit{left\text{-}bor}(R) = \mathit{bor}_R$ and $\mathit{right\text{-}bor}(R) = \mathit{right\text{-}bor}(V)$.
        Let $\mathcal{I}_L^{\mathtt{B}}$ be the instance $(D[L], b, s, \mathit{first}, \mathit{mid}, \mathit{left\text{-}bor}(L), \mathit{right\text{-}bor}(L), \mathtt{B}_{in}^{newL})$.
        Let $\mathcal{I}_R^{\mathtt{B}}$ be the instance $(D[R], b, s, \mathit{mid}, \mathit{last}, \mathit{left\text{-}bor}(R), \mathit{right\text{-}bor}(R), \mathtt{B}_{in}^{newR})$.
        if $\mathcal{I}_L^{\mathtt{B}}$ and $\mathcal{I}_R^{\mathtt{B}}$ are legitimate inputs for Algorithm 1 then
            if Algorithm $1(\mathcal{I}_L^{\mathtt{B}})$ != NO and Algorithm $1(\mathcal{I}_R^{\mathtt{B}})$ != NO then
                return Algorithm 1 $(\mathcal{I}_L^{\mathtt{B}}) \cup$ Algorithm 1 $(\mathcal{I}_R^{\mathtt{B}})$
return NO
```

For the formal description of Algorithm 1 refer to the pseudocode. We give the informal description of Algorithm 1 here. In a legitimate instance when the number of vertices is $b \cdot \delta$, the number of buckets is δ. Recall that $\delta = \lceil {}^{s+1}/_{b} \rceil$. Note that in this case, every b-bucketing of the vertex set has external stretch at most s. This is because the number of buckets between any two buckets is at most $\delta - 2$ and hence, the number of vertices that appear in the buckets between any two buckets is at most $(\delta - 2) \cdot b \leq s$.

When the number of vertices is larger, the algorithm first guesses which vertices will be assigned a bucket from the first half buckets of the final bucketing (this corresponds to the set L) and which will be assigned the last half (this corresponds to the set R). Since the final bucketing has to be consistent with $\mathtt{B}_{in}$, from the description of $\mathtt{B}_{in}$ in Definition 5.2, the vertices of $\mathit{left}\text{-}\mathit{bor}(V)$ should belong to the first half buckets and the vertices of $\mathit{right}\text{-}\mathit{bor}(V)$ should belong to the last half buckets. Thus, the algorithm only considers those partitions (guesses) which $\mathtt{B}_{in}$ validates (Line 5).

Fix a guessed partition (L, R) of V. The set bor_L represents the set of vertices in L that have an out-neighbour in R. Similarly, the set bor_R represents the set of vertices in R with an in-neighbour in L. Since in any b-bucketing of external stretch at most s, the number of buckets that can appear between the buckets of the end points of a forward arc is at most $\delta - 1$, the vertices of bor_L can only be placed in the δ buckets closest to the middle bucket and before it. Similarly, the vertices of bor_R can only be placed in the δ buckets closest to the middle bucket and after it. The algorithm goes over all possible partial b-bucketings of these vertices in the described buckets, that are consistent with $\mathtt{B}_{in}$, and themselves have external stretch at most δ (Line 8).

For a fixed partial b-bucketing enumerated above, the algorithm recursively finds a bucketing of the L vertices in the first half buckets and the bucketing of the R vertices in the last half buckets that is consistent with $\mathtt{B}_{in}$ and the partial b-bucketing of the bor_L and bor_R vertices guessed. This final bucketing is then obtained by combing the two bucketings from the two disjoint sub-problems (Lines 9 to 19).

▶ **Lemma 5.5.** ($\star$) Algorithm 1 on a legitimate input $(D, b, s, first, last, \mathit{left}\text{-}\mathit{bor}(V), \mathit{right}\text{-}\mathit{bor}(V), B_{in})$, runs in time $\min\{\mathcal{O}^*(4^n \cdot \lceil {s+1}/{b} \rceil^n), \mathcal{O}^*(4^n \cdot \lceil {s+1}/{b} \rceil^{2(b+s)\log n})\}$, and returns a look-out bucketing for $\mathcal{I}$, if it exists.

Theorem 1.3 (resp. Theorem 1.3) can be proved by setting bucket size to be $\lceil {b\epsilon}/{2} \rceil$ (resp. 1) and external stretch $b - 1$ as parameters in the algorithm of Lemma 5.1. The full proofs are deferred to the full version.

6 Exact Algorithm for Digraph Bandwidth via Directed Homomorphisms

The goal of this section is to prove Theorem 1.5. Towards this, we give a reduction from Digraph Bandwidth to a subgraph homomorphism problem for digraphs. Given two digraphs D and H, a *directed homomorphism from D to H* is a function $h : V(D) \to V(H)$, such that if $(u, v) \in E(D)$, then $(h(u), h(v)) \in E(H)$. A directed homomorphism that is injective is called an *injective directed homomorphism.* For any positive integers n, b such that $b \leq n$, we denote by P_n^b the directed graph on n vertices such that $V(P_n^b) = [n]$ and $E(P_n^b) = E_f \uplus E_b$, where $E_b = \{(i, j) : i > j, i, j \in [n]\}$ and $E_f = \{(i, i+j) : i \in [n-1], j \in [b]\}$. In the following lemma, we build the relation between Digraph Bandwidth of D and injective homomorphism from D to P_n^b.

▶ **Lemma 6.1.** *For any digraph D and an integer b, D has bandwidth at most b if and only if there is an injective homomorphism from D to P_n^b.*

Proof. In the forward direction, suppose that D has digraph bandwidth at most b. Let σ be a b-ordering of D. Let $f \colon V(D) \to V(P_n^b)$ be a function such that $f(u) = \sigma(u)$. We claim that f is an injective homomorphism. Since σ is an ordering of D, f is an injective function. We prove that it is also a homomorphism. Consider an arc $(u, v) \in E(D)$. If $\sigma(u) < \sigma(v)$, then since σ is a b-ordering, $\sigma(v) - \sigma(u) \leq b$. Therefore, $(f(u), f(v)) \in E_f$. If $\sigma(u) > \sigma(v)$, then $(f(u), f(v)) \in E_b$. Hence, $(f(u), f(v)) \in E(P_n^b)$. In the backward direction, suppose that there exists an injective homomorphism from D to P_n^b. Let $f : V(D) \to V(P_n^b)$ be a function. Then, we claim that $\sigma = (f^{-1}(1), \cdots, f^{-1}(n))$ is a b-ordering of D. Suppose not, then there exists an arc $(u, v) \in E(D)$ such that $\sigma(v) - \sigma(u) > b$. Let $u = f^{-1}(j)$ and $v = f^{-1}(k)$. Note that $\sigma(u) = j$ and $\sigma(v) = k$. Therefore, $j < k$. Since $k - j > b$, $(j, k) \notin E(P_n^b)$, a contradiction that f is an injective homomorphism. ◀

For any two digraphs D and H, let $inj(D, H)$ denote the number of injective homomorphisms from D to H and let $hom(D, H)$ denote the number of homomorphisms from D to H. Then the following lemma holds from Theorem 1 in [1].

▶ **Lemma 6.2** (Theorem 1, [1]). *Suppose that D and H be two digraphs such that $|V(D)| = |V(H)|$. Then,*

$$inj(D, H) = \sum_{W \subseteq V(D)} (-1)^{|W|} hom(D \setminus W, H).$$

Now, we state the following known result about the number of homomorphisms between two given digraphs D and H.

▶ **Lemma 6.3** (Theorem 3.1, 5.1 [11]). *Given digraphs D and H together with a tree decomposition of D of width $\texttt{tw}$, $hom(D, H)$ can be computed in time $\mathcal{O}(nh^{\texttt{tw}+1} \min\{\texttt{tw}, h\})$, where $n = |V(D)|$ and $h = |V(H)|$.*

Using Lemmas 6.2 and 6.3, we get the following result.

▶ **Lemma 6.4.** *Given digraphs D and H together with a tree decomposition of D of width $\texttt{tw}$, $inj(D, H)$ can be computed in time $\mathcal{O}(2^n nh^{\texttt{tw}+1} \min\{\texttt{tw}, h\})$, where $n = |V(D)|$ and $h = |V(H)|$.*

Proof of Theorem 1.5. The proof follows from Lemmas 6.1 and 6.4. ◀

7 Conclusion

In this paper we gave exponential time algorithm for the Digraph Bandwidth problem, that either solved the problem exactly or computed it approximately. In particular, our results imply that whenever $b \leq \frac{n}{\log^2 n}$ or, the treewidth of the underlying undirected digraph is $\mathcal{O}(\frac{n}{\log n})$ or, the number of arcs in the digraph are linear in the number of vertices, then there exists a $2^{\mathcal{O}(n)}$ time algorithm for solving Digraph Bandwidth. Some important questions that remain open about Digraph Bandwidth are the following.

- Does Digraph Bandwidth admit an algorithm with running time $2^{\mathcal{O}(n)}$ on general digraphs?
- Another interesting question is the complexity of the Digraph Bandwidth problem, when b is fixed. Recall that, in the undirected case, Bandwidth can be solved in time $\mathcal{O}(n^{b+1})$ [26]. When $b = 0$, the problem is equivalent to checking if the input is a dag, which can be done in linear time. For $b = 1$, we are able to design an $\mathcal{O}(n^2)$ time algorithm. For $b = 2$, the problem seems to be extremely complex, and in fact, we will be surprised if the problem turns out to be polynomial time solvable. Overall, finding the complexity of Digraph Bandwidth, for a fixed $b \geq 2$, is an interesting open problem.

References

1 O. Amini, F.V. Fomin, and S. Saurabh. Counting Subgraphs via Homomorphisms. *SIAM J. Discrete Math.*, 26(2):695–717, 2012.

2 S.F. Assmann, G.W. Peck, M.M. Sysło, and J. Zak. The bandwidth of caterpillars with hairs of length 1 and 2. *SIAM J. Alg. Discrete Meth.*, 2(4):387–393, 1981.

3 G. Blache, M. Karpiński, and J. Wirtgen. *On approximation intractability of the bandwidth problem.* Citeseer, 1997.

4 H.L. Bodlaender, M.R. Fellows, and M.T. Hallett. Beyond NP-completeness for problems of bounded width (extended abstract): hardness for the W hierarchy. In *Proc. of STOC 1994*, pages 449–458, 1994.

5 M. Chudnovsky, A. Fradkin, and P. Seymour. Tournament Immersion and Cutwidth. *J. Comb. Theory Ser. B*, 102(1):93–101, 2012.

6 M. Cygan and M. Pilipczuk. Faster Exact Bandwidth. In *Proc. of WG 2008*, pages 101–109, 2008.

7 M. Cygan and M. Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*, 411(40-42):3701–3713, 2010.

8 M. Cygan and M. Pilipczuk. Bandwidth and distortion revisited. *Discrete Appl. Math.*, 160(4-5):494–504, 2012.

9 M. Cygan and M. Pilipczuk. Even Faster Exact Bandwidth. *ACM Trans. Algorithms*, 8(1):8:1–8:14, 2012.

10 Marek Cygan and Marcin Pilipczuk. Faster exact bandwidth. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 101–109. Springer, 2008.

11 J. Díaz, M. Serna, and D.M. Thilikos. Counting H-colorings of partial k-trees. *Theor. Comput. Sci.*, 281(1-2):291–309, 2002.

12 M.S. Dregi and D. Lokshtanov. Parameterized Complexity of Bandwidth on Trees. In *Proc. of ICALP 2014*, pages 405–416, 2014.

13 U. Feige. Coping with the NP-hardness of the graph bandwidth problem. In *Proc. of SWAT 2000*, pages 10–19. Springer, Berlin, 2000.

14 U. Feige and K. Talwar. Approximating the Bandwidth of Caterpillars. In *Proc. of APPROX-RANDOM 2005*, volume 3624, pages 62–73, 2005.

15 M. Fürer, S. Gaspers, and S.P. Kasiviswanathan. An exponential time 2-approximation algorithm for bandwidth. *Theor. Comput. Sci.*, 511:23–31, 2013.

16 M. Garey, R. Graham, D. Johnson, and D. Knuth. Complexity Results for Bandwidth Minimization. *SIAM J. Appl. Math.*, 34(3):477–495, 1978.

17 M.R. Garey and D.S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

18 P.A. Golovach, P. Heggernes, D. Kratsch, D. Lokshtanov, D. Meister, and S. Saurabh. Bandwidth on AT-free graphs. *Theor. Comput. Sci.*, 412(50):7001–7008, 2011.

19 P. Heggernes, D. Kratsch, and D. Meister. Bandwidth of bipartite permutation graphs in polynomial time. *J. Discrete Algorithms*, 7(4):533–544, 2009.

20 D.J. Kleitman and R. Vohra. Computing the Bandwidth of Interval Graphs. *SIAM J. Discrete Math.*, 3(3):373–375, 1990.

21 R. Krauthgamer, J.R. Lee, M. Mendel, and A. Naor. Measured Descent: A New Embedding Method for Finite Metrics. In *Proc. of FOCS 2004*, pages 434–443, 2004.

22 Yung-Ling Lai and Kenneth Williams. A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. *Journal of graph theory*, 31(2):75–94, 1999.

23 Marek M. Karpinski, Jürgen Wirtgen, and A. Zelikovsky. An Approximation Algorithm for the Bandwidth Problem on Dense Graphs. Technical report, University of Bonn, 1997.

24 B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM J. Alg. Discrete Meth.*, 7(4):505–512, 1986.

25 C.H. Papadimitriou. The NP-Completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

26 J.B. Saxe. Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM J. Alg. Discrete Meth.*, 1(4):363–369, 1980.

27 W. Unger. The complexity of the approximation of the bandwidth problem. In *Proc. of FOCS 1998*, pages 82–91, 1998.

28 V. Vassilevska, R. Williams, and S.L.M. Woo. Confronting hardness using a hybrid approach. In *Proc. of SODA*, pages 1–10, 2006.

29 J.H. Yan. The bandwidth problem in cographs. *Tamsui Oxford J. Math. Sci*, 13:31–36, 1997.

An $O(n^{1/4+\epsilon})$ Space and Polynomial Algorithm for Grid Graph Reachability

Rahul Jain
Indian Institute of Technology Kanpur, India
jain@cse.iitk.ac.in

Raghunath Tewari
Indian Institute of Technology Kanpur, India
rtewari@cse.iitk.ac.in

Abstract

The reachability problem is to determine if there exists a path from one vertex to another in a graph. Grid graphs are the class of graphs where vertices are present on the lattice points of a two-dimensional grid, and an edge can occur between a vertex and its immediate horizontal or vertical neighbor only.

Asano et al. presented the first simultaneous time space bound for reachability in grid graphs by presenting an algorithm that solves the problem in polynomial time and $O(n^{1/2+\epsilon})$ space. In 2018, the space bound was improved to $\tilde{O}(n^{1/3})$ by Ashida and Nakagawa.

In this paper, we show that reachability in an n vertex grid graph can be decided by an algorithm using $O(n^{1/4+\epsilon})$ space and polynomial time simultaneously.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases graph reachability, grid graph, graph algorithm, sublinear space algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.19

Funding *Rahul Jain*: Ministry of Human Resource Development, Government of India
Raghunath Tewari: DST Inspire Faculty Grant, Visvesvaraya Young Faculty Fellowship

Acknowledgements We thank anonymous reviewers for their helpful comments on an earlier version of this paper.

1 Introduction

The problem of graph reachability is to decide whether there is a path from one vertex to another in a given graph. This problem has several applications in the field of algorithms and computational complexity theory. Reachability in directed and undirected graphs capture the complexity of nondeterministic and deterministic logarithmic space respectively [12]. It is often used as a subroutine in various network related problems. Hence designing better algorithms for this problem is of utmost importance to computer scientists.

Standard graph traversal algorithms such as DFS and BFS give a linear time algorithm for this problem, but they require linear space as well. Savitch's divide and conquer based algorithm can solve reachability in $O(\log^2 n)$ space, but as a tradeoff, it requires $n^{O(\log n)}$ time [13]. Hence it is natural to ask whether we can get the best of both worlds and design an algorithm for graph reachability that runs in polynomial time and uses polylogarithmic space. Wigderson asked a relaxed version of this question in his survey - whether graph reachability can be solved by an algorithm that runs simultaneously in polynomial time and uses $O(n^{1-\epsilon})$ space [15]. In this paper, we address this problem for a certain restricted class of directed graphs.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 19; pp. 19:1–19:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Barnes et al showed that reachability in general graphs can be decided simultaneously in $n/2^{\Theta(\sqrt{\log n})}$ space and polynomial time [6]. Although this algorithm gives a sublinear space bound, it still does not give a positive answer to Wigderson's question.

However, for certain topologically restricted graph classes, we know of better space bounds simultaneously with polynomial time. Planar graphs are graphs that can be drawn on the plane such that no two edges of the graph cross each other at an intermediate point. Imai et al. showed that reachability in planar graph can be solved in $O(n^{1/2+\epsilon})$ space for any $\epsilon > 0$ [9] . Later this space bound was improved to $\tilde{O}(n^{1/2})$ [4]. For graphs of higher genus, Chakraborty et al. gave an $\tilde{O}(n^{2/3}g^{1/3})$ space algorithm which additionally requires an embedding of the graph on a surface of genus g, as input [7]. They also gave an $\tilde{O}(n^{2/3})$ space algorithm for H minor-free graphs which requires tree decomposition of the graph as an input and $O(n^{1/2+\epsilon})$ space algorithm for $K_{3,3}$-free and K_5-free graphs. For layered planar graphs, Chakraborty and Tewari showed that for every $\epsilon > 0$ there is an $O(n^\epsilon)$ space algorithm [8]. Stolee and Vinodchandran presented a polynomial time algorithm that, for any $\epsilon > 0$ solves reachability in a directed acyclic graph with $O(n^\epsilon)$ sources and embedded on the surface of genus $O(n^\epsilon)$ using $O(n^\epsilon)$ space [14]. For unique-path graphs, Kannan et al. presented a $O(n^\epsilon)$ space and polynomial time algorithm [10].

Grid graphs are a subclass of planar graphs whose vertices are present at the integer lattice points of an $m \times m$ grid and edges can only occur between a vertex and its immediate vertical or horizontal neighbor. It was known that reachability in planar graphs can be reduced to reachability in grid graphs in logarithmic space [1]. The reduction, however, causes atleast a quadratic blow-up in size with respect to the input graph. In this paper, we study the simultaneous time-space complexity of reachability in grid graphs.

Asano and Doerr presented a polynomial time algorithm that uses $O(n^{1/2+\epsilon})$ space for solving reachability in grid graphs [3]. Ashida and Nakagawa presented an algorithm with improved space complexity of $\widetilde{O}(n^{1/3})$ [5]. The latter algorithm proceeded by first dividing the input grid graph into subgrids. It then used a gadget to transform each subgrid into a planar graph, making the whole of the resultant graph planar. Finally, it used the planar reachability algorithm of Imai et al. [9] as a subroutine to get the desired space bound.

In this paper, we present a $O(n^{1/4+\epsilon})$ space and polynomial time algorithm for grid graph reachability, thereby significantly improving the space bound of Ashida and Nakagawa.

▶ **Theorem 1** (Main Theorem). *For every $\epsilon > 0$, there exists a polynomial time algorithm that can solve reachability in an n vertex grid graph, using $O(n^{1/4+\epsilon})$ space.*

To solve the problem we divide the given grid graph into subgrids and replace paths in each grid with a single edge between the boundary vertices to get an *auxiliary graph*. This reduces the size of the graph and preserves reachability. Instead of trying to convert the auxiliary graph into planar graph while preserving reachability (which was the approach of Ashida and Nakagawa [5]), we use a divide and conquer strategy to directly solve reachability problem in the auxiliary graph. We define and use a new type of graph separator of the auxiliary graph, that we call as *pseudoseparator* and use it to divide the auxiliary graph into small components and then combine the solution in a space-efficient manner.

In Section 2 we state the definitions and notations that we use in this paper. In Section 3 we define the auxiliary graph and state various properties of it that we use later. In Section 4 we discuss the concept of a pseudoseparator. We give its formal definition and show how a pseudoseparator can be computed efficiently. In Section 5 we give the algorithm to solve reachability in an auxiliary graph and prove its correctness. Finally in Section 6 we use the algorithm of Section 5 to give an algorithm to decide reachability in grid graphs and thus prove Theorem 1.

2 Preliminaries

Let $[n]$ denote the set $\{0, 1, 2, \ldots, n\}$. We denote the vertex set of a graph G by $V(G)$ and its edge set by $E(G)$. For a subset U of $V(G)$, we denote the subgraph of G induced by the vertices of U as $G[U]$. For a graph G, we denote $\mathsf{cc}(G)$ as the set of all connected components in the underlying undirected graph of G, where the undirected version is obtained by removing orientations on all edges of G. Henceforth, whenever we talk about connected components, we will mean the connected components of the underlying undirected graph.

In a *drawing* of a graph on a plane, each vertex is mapped to a point on the plane, and each edge is mapped to a simple arc whose endpoints coincide with the mappings of the end vertices of the edge. Moreover, the interior points of an arc corresponding to an edge does not intersect with any other vertex points. A graph is said to be *planar* if it can be drawn on the plane such that no two edges of the graph intersect except possibly at the endpoints. We will not be concerned with details about the representation of planar graphs. We note that the work of [2], and subsequently [12], implies a deterministic logarithmic space algorithm that decides whether or not a given graph is planar, and if it is, outputs a planar embedding. Hence when dealing with planar graphs, we will assume without loss of generality that a planar embedding is provided as well.

A $m \times m$ *grid graph* is a directed graph whose vertices are $[m] \times [m] = \{0, 1, \ldots, m\} \times \{0, 1, \ldots, m\}$ so that if $((i_1, j_1), (i_2, j_2))$ is an edge then $|i_1 - i_2| + |j_1 - j_2| = 1$. It follows from definition that grid graphs are a subset of planar graphs.

3 Auxiliary Graph

Let G be an $m \times m$ grid graph. We divide G into $m^{2\alpha}$ subgrids such that each subgrid is a $m^{1-\alpha} \times m^{1-\alpha}$ grid. Formally, for $1 \leq i, j \leq m^{\alpha}$, the (i, j)-th *subgrid* of G, denoted as $G[i, j]$ is the subgraph of G induced by the set of vertices, $V(G[i, j]) = \{(i', j') \mid (i-1) \cdot m^{1-\alpha} \leq i' \leq i \cdot m^{1-\alpha}$ and $(j-1) \cdot m^{1-\alpha} \leq j' \leq j \cdot m^{1-\alpha}\}$.

For $0 < \alpha < 1$ and $1 \leq i, j \leq m^{\alpha}$, we define $\mathsf{Aux}_{\alpha}(G)[i, j]$ as follows. The vertex set of $\mathsf{Aux}_{\alpha}(G)[i, j]$ is $V(\mathsf{Aux}_{\alpha}(G)[i, j]) = \{(i', j') \mid i' = k \cdot m^{1-\alpha}$ or $j = l \cdot m^{1-\alpha}$, such that $k \in \{i-1, i\}$ and $l \in \{j-1, j\}\}$. For two vertices u, v in $\mathsf{Aux}_{\alpha}(G)[i, j]$, (u, v) is an edge in $\mathsf{Aux}_{\alpha}(G)[i, j]$ if there is a path from u to v in the subgrid $G[i, j]$. In a drawing of $\mathsf{Aux}_{\alpha}(G)[i, j]$, we use a straight line to represent the edge if u and v do not lie on a single side of $\mathsf{Aux}_{\alpha}(G)[i, j]$, and an arc present inside the grid to represent it otherwise.

Now for $0 < \alpha < 1$, we define the *α-auxiliary graph*, $\mathsf{Aux}_{\alpha}(G)$ as follows. The vertex set of $\mathsf{Aux}_{\alpha}(G)$, $V(\mathsf{Aux}_{\alpha}(G)) = \{(i, j) \mid i = k \cdot m^{1-\alpha}$ or $j = l \cdot m^{1-\alpha}$, such that $0 \leq k, l \leq m^{\alpha}\}$. The edges of $\mathsf{Aux}_{\alpha}(G)$ are the edges of $\mathsf{Aux}_{\alpha}(G)[i, j]$ taken over all pairs (i, j). Note that $\mathsf{Aux}_{\alpha}(G)$ might have parallel edges, since an edge on a side of a block might be present in the adjacent block as well. In such cases we preserve both the edges, however in their different blocks of $\mathsf{Aux}_{\alpha}(G)$ in the drawing of $\mathsf{Aux}_{\alpha}(G)$ on the plane. Figure 1 contains an example of a grid graph partitioned into subgrids and its corresponding auxiliary graph. Since each block $\mathsf{Aux}_{\alpha}(G)[i, j]$ contains $4m^{1-\alpha}$ vertices, the total number of vertices in $\mathsf{Aux}_{\alpha}(G)$ would be at most $4m^{1+\alpha}$.

Our algorithm for reachability first constructs $\mathsf{Aux}_{\alpha}(G)$ by solving each of the $m^{1-\alpha} \times m^{1-\alpha}$ grids recursively. It then uses a polynomial time subroutine to decide reachability in $\mathsf{Aux}_{\alpha}(G)$. Note that we do not store the graph $\mathsf{Aux}_{\alpha}(G)$ explicitly, since that would require too much space. Rather we solve a subgrid recursively whenever the subroutine queries for an edge in that subgrid of $\mathsf{Aux}_{\alpha}(G)$.

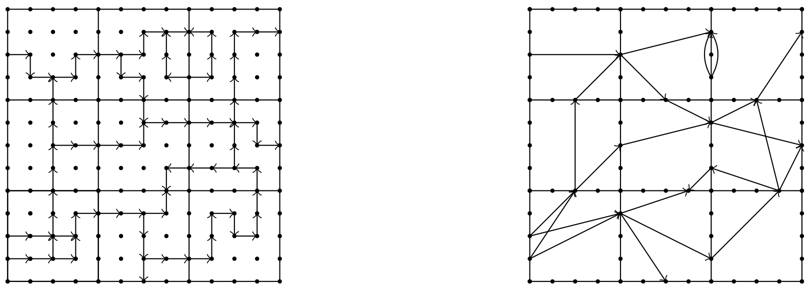


Figure 1 A grid graph G divided into subgrids and its corresponding auxiliary graph $\mathsf{Aux}_\alpha(G)$.

Our strategy is to develop a polynomial time algorithm which solves reachability in $\mathsf{Aux}_\alpha(G)$ using $\tilde{O}(\tilde{m}^{1/2+\beta/2})$ space where $\tilde{m}$ is the number of vertices in $\mathsf{Aux}_\alpha(G)$. As discussed earlier, $\tilde{m}$ can be at most $4m^{1+\alpha}$. Hence, the main algorithm would require $\tilde{O}(m^{1/2+\beta/2+\alpha/2+\alpha\beta/2})$ space. For a fixed constant $\epsilon > 0$, we can pick $\alpha > 0$ and $\beta > 0$ such that the space complexity becomes $O(m^{1/2+\epsilon})$.

3.1 Properties of the Auxiliary Graph

In the following definition, we give an ordered labeling of the vertices on some block of the auxiliary graph. The labeling is defined with respect to some vertex present in the block.

▶ **Definition 2.** *Let G be a $m \times m$ grid graph, $l = \mathsf{Aux}_\alpha(G)[i,j]$ be a block of $\mathsf{Aux}_\alpha(G)$ and $v = (x, y)$ be a vertex in $\mathsf{Aux}_\alpha(G)[i,j]$. Let $t = m^{1-\alpha}$. We define the counter-clockwise adjacent vertex of v with respect to the block l, $c_l(v)$ as follows:*

$$c_l(v) = \begin{cases} (x+1, y) & \text{if } x < (i+1)t \text{ and } y = jt \\ (x, y+1) & \text{if } x = (i+1)t \text{ and } y < (j+1)t \\ (x-1, y) & \text{if } x > it \text{ and } y = (j+1)t \\ (x, y-1) & \text{if } x = it \text{ and } y > jt \end{cases}$$

Similarly we also define the r-th counter-clockwise adjacent neighbour of v with respect to the block l inductively as follows. For $r = 0$, $c_l^r(v) = v$ and otherwise we have $c_l^{r+1}(v) = c_l(c_l^r(v))$.

Note that for a block l and vertices v and w in it, we write v as $c_l^p(w)$ where p is smallest non-negative integer for which $c_l^p(w) = v$. Next we formalize what it means to say that two edges of the auxiliary graph cross each other.

▶ **Definition 3.** *Let G be a grid graph and l be a block of $\mathsf{Aux}_\alpha(G)$. For two distinct edges e and f in the block, such that $e = (v, c_l^p(v))$ and $f = (c_l^q(v), c_l^r(v))$. We say that edges e and f cross each other if $\min(q,r) < p < \max(q,r)$.*

Note the definition of cross given above is symmetric. That is, if edges e and f cross each other then f and e must cross each other as well. For an edge $f = (c_l^q(v), c_l^r(v))$, we define $\overleftarrow{f} = (c_l^r(v), c_l^q(v))$ and call it the *reverse* of f. We also note that if e and f cross each other, then e and $\overleftarrow{f}$ also cross each other.

In Lemma 4 we state an equivalent condition of crossing of two edges, and in Lemmas 6 and 7 we state certain properties of the auxiliary graph that we use later.

▶ **Lemma 4.** *Let G be a grid graph and l be a block of $\mathsf{Aux}_\alpha(G)$. Let w be an arbitrary vertex in the block l and $e = (c_l^p(w), c_l^q(w))$ and $f = (c_l^r(w), c_l^s(w))$ be two distinct edges in l. Then e and f cross each other if and only if either of the following two conditions hold:*

- $\min(p,q) < \min(r,s) < \max(p,q) < \max(r,s)$
- $\min(r,s) < \min(p,q) < \max(r,s) < \max(p,q)$

Proof. We prove that if $\min(p,q) < \min(r,s) < \max(p,q) < \max(r,s)$ then e and f cross each other. We let $p < r < q < s$. Other cases can be proved by reversing appropriate edges. We thus have integers $r_1 = r - p$, $q_1 = q - p$ and $s_1 = s - p$. Clearly, $r_1 < q_1 < s_1$. Let $v = c_l^p(w)$. Thus we have $e = (v, c_l^q(w)) = (v, c_l^{q_1}(c_l^p(w))) = (v, c_l^{q_1}(v))$ and $f = (c_l^r(w), c_l^s(w)) = (c_l^{r_1}(c_l^p(w)), c_l^{s_1}(c_l^p(w))) = (c_l^{r_1}(v), c_l^{s_1}(v))$ The proof for the second condition is similar.

Now, we prove that if $e = (c_l^p(w), c_l^q(w))$ and $f = (c_l^r(w), c_l^s(w))$ cross each other then either of the given two condition holds. We assume that p is the smallest integer among p, q, r and s. Other cases can be proved similarly. Now, let $v = c_l^p(w)$. We thus have integers $q_1 = q - p, r_1 = r - p$ and $s_1 = s - p$ such that $e = (v, c_l^{q_1}(v))$ and $f = (c_l^{r_1}(v), c_l^{s_1}(v))$. Since e and f cross each other, we have $\min(r_1, s_1) < q_1 < \max(r_1, s_1)$. Thus $\min(r_1 + p, s_1 + p) < q_1 + p < \max(r_1 + p, s_1 + p)$. It follows that $\min(r,s) < q < \max(r,s)$. Since we assumed p to be smallest integer among p, q, r and s; we have $\min(p,q) < \min(r,s) < \max(p,q) < \max(r,s)$, thus proving the lemma. ◀

We see that we can draw an auxiliary graph on a plane such that the arcs corresponding to two of its edges intersect if and only if the corresponding edges cross each other. Henceforth, we will work with such a drawing.

▶ **Definition 5.** *Let G be a grid graph and l be a block of $\mathsf{Aux}_\alpha(G)$. For a vertex v and edges f, g such that $f = (c_l^q(v), c_l^r(v))$ and $g = (c_l^s(v), c_l^t(v))$, we say that f is closer to v than g if $\min(q,r) < \min(s,t)$.*

We say f is closest *to v if there exists no other edge f' which is closer to v than f.*

▶ **Lemma 6.** *Let G be a grid graph and $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be two edges in $\mathsf{Aux}_\alpha(G)$. If e_1 and e_2 cross each other, then $\mathsf{Aux}_\alpha(G)$ also contains the edges (u_1, v_2) and (u_2, v_1).*

Proof. Let $e_1 = (v, c_l^p(v))$ and $e_2 = (c_l^q(v), c_l^r(v))$ be two edges that cross each other in $\mathsf{Aux}_\alpha(G)$. Let l be the block of $\mathsf{Aux}_\alpha(G)$ to which e_1 and e_2 belong. Consider the subgrid of G which is solved to construct the block l. Since the edge e_1 exists in block l, there exists a path P from v to $c_l^p(v)$ in the underlying subgrid. This path P divides the subgrid into two parts such that the vertices $c_l^q(v)$ and $c_l^r(v)$ belong to different parts of the subgrid. Thus, a path between $c_l^q(v)$ and $c_l^r(v)$ necessarly take a vertex of path P. Hence, there is a path from v to $c_l^r(v)$ and a path from $c_l^p(v)$ to $c_l^r(v)$. Thus the lemma follows. ◀

▶ **Lemma 7.** *Let G be a grid graph and $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be two edges in $\mathsf{Aux}_\alpha(G)$. If e_1 and e_2 cross a certain edge $f = (x, y)$, and e_1 is closer to x than e_2, then the edge (u_1, v_2) is also present in $\mathsf{Aux}_\alpha(G)$.*

Proof. Let $e = (v, c_l^p(v))$, $f = (c_l^q(v), c_l^r(v))$ and $g = (c_l^s(v), c_l^t(v))$. If $c_l^q(v) = c_l^s(v)$ then the lemma trivially follows. Otherwise, we have two cases to consider:

Case 1 (f crosses g): In this case, we will have $(c_l^q(v), c_l^t(v))$ present in $\mathsf{Aux}_\alpha(G)$ by Lemma 6.

Case 2 (f does not cross g): In this case, we have $\min(q,r) < \min(s,t) < p < \max(s,t) < \max(q,r)$. Since f crosses e, we have the edge $(c_l^q(v), c_l^p(v))$ in $\mathsf{Aux}_\alpha(G)$ by Lemma 6. This edge will cross g. Hence $(c_l^q(v), c_l^t(v))$ is present in $\mathsf{Aux}_\alpha(G)$. ◀

4 Pseudoseparator in a Grid Graph

Imai et al. used a separator construction to solve the reachability problem in planar graphs [9]. A separator is a set of vertices whose removal disconnects the graph into *small components*. An essential property of a separator is that, for any two vertices, a path between the vertices must contain a separator vertex if the vertices lie in two different components with respect to the separator.

Grid graphs are subclasses of planar and are known to have good separators. However, for a grid graph G, the graph $\mathsf{Aux}_\alpha(G)$ might not have a small separator. Here we show that $\mathsf{Aux}_\alpha(G)$ has a different kind of separator, which we call as a PseudoSeparator (see Definition 8). PseudoSeparator allows us to decide reachability in $\mathsf{Aux}_\alpha(G)$, by using a divide and conquer strategy and obtain the claimed time and space bounds.

For a graph $H = (V_1, E_1)$ given along with its drawing, and a subgraph $C = (V_2, E_2)$ of H, define the graph $H \diamond C = (V_3, E_3)$ as $V_3 = V_1 \setminus V_2$ and $E_3 = E_1 \setminus \{e \in E_1 \mid \exists f \in E_2, e \text{ crosses } f\}$. We note that the graph H we will be working with throughout the article will be a subgraph of an auxiliary graph. Hence it will always come with a drawing.

▶ **Definition 8.** *Let G be a grid graph and H be a vertex induced subgraph of $\mathsf{Aux}_\alpha(G)$ with h vertices. Let $f : \mathbb{N} \to \mathbb{N}$ be a function. A subgraph C of H is said to be an $f(h)$-*PseudoSeparator *of $\mathsf{Aux}_\alpha(G)$ if the size of every connected component in $\mathsf{cc}(H \diamond C)$ is at most $f(h)$. The size of C is the total number of vertices and edges of C summed together.*

For a vertex-induced subgraph H of $\mathsf{Aux}_\alpha(G)$, an $f(h)$-PseudoSeparator is a subgraph C of H that has the property that, if we remove the vertices as well as all the edges that cross one of the edges of the PseudoSeparator, the graph gets disconnected into small pieces. Moreover for every edge e in H, if there exists distinct sets U_1 and U_2 in $\mathsf{cc}(H \diamond C)$ such that one of the endpoints of e is in U_1 and the other is in U_2, then there exists an edge f in C such that e crosses f. Hence any path which connects two vertices in different components, must either contain a vertex of C or must contain an edge that crosses an edge of C. We divide the graph using this PseudoSeparator and give an algorithm which recursively solves each subgraph and then combines their solution efficiently.

4.1 Constructing a Pseudoseparator

We briefly comment on how to construct a PseudoSeparator of a vertex induced subgraph H of $\mathsf{Aux}_\alpha(G)$. First, we pick a maximal subset of edges from H so that no two edges cross (see Defintion 9). Then we triangulate the resulting graph. This can be done in logspace. Next, we use Imai et al.'s algorithm to find a separator of the triangulated graph. Call the triangulated graph as $\widehat{H}$ and the separator vertices as S. The vertex set of PseudoSeparator of H will contain all the vertices of S and four additional vertices for each edge of $\widehat{H}[S]$ that is not present in H. The edge set of PseudoSeparator of H will contain all the edges of H which are also in $\widehat{H}[S]$ and four additional edges for each edge of $\widehat{H}[S]$ that is not present in H.

▶ **Definition 9.** *Let G be a grid graph and H be a vertex induced subgraph of $\mathsf{Aux}_\alpha(G)$. We define $\mathsf{planar}(H)$ as a subgraph of H. The vertex set of $\mathsf{planar}(H)$ is same as that of H. For an edge $e \in H$, let l be the block to which e belongs and let w be the lowest indexed vertex in that block. Then $e = (c_l^i(w), c_l^j(w))$ is in $\mathsf{planar}(H)$ if there exists no other edge $f = (c_l^x(w), c_l^y(w))$ in H such that $\min(x, y) < \min(i, j) < \max(x, y) < \max(i, j)$.*

In Lemma 10 we show that the graph $\mathsf{planar}(H)$ is indeed planar and prove a simple yet crucial property of this graph, that would help us to construct the PseudoSeparator.

▶ **Lemma 10.** *Let G be a grid-graph and H be a vertex induced subgraph of $\mathsf{Aux}_\alpha(G)$. No two edges of $\mathsf{planar}(H)$ cross each other. Moreover, for any edge e in H that is not in $\mathsf{planar}(H)$, there exists another edge in $\mathsf{planar}(H)$ that crosses e.*

Proof. Let l be a block of $\mathsf{Aux}_\alpha(G)$ and w be the smallest index vertex of l. Let $e = (c_l^p(w), c_l^q(w))$ and $f = (c_l^r(w), c_l^s(w))$ be two edges of H that cross. We have, by lemma 4, that either $\min(p,q) < \min(r,s) < \max(p,q) < \max(r,s)$ or $\min(r,s) < \min(p,q) < \max(r,s) < \max(p,q)$. Hence, by our construction of $\mathsf{planar}(H)$, atmost one of e and f belongs to it. Thus no two edges of $\mathsf{planar}(H)$ can cross.

For the second part, we will prove by contradiction. Let us assume that there exists edges in H which is not in $\mathsf{planar}(H)$ and also not crossed by an edge in $\mathsf{planar}(H)$. We pick edge $e = (c_l^p(w), c_l^q(w))$ from them such that $\min(p,q)$ of that edge is minimum. Since this edge is not present in $\mathsf{planar}(H)$, we have by definition, an edge $f = (c_l^r(w), c_l^s(w))$ such that $\min(r,s) < \min(p,q) < \max(r,s) < \max(p,q)$. We pick the edge f for which $\min(r,s)$ is minimum. Now, since this edge f is not present in $\mathsf{planar}(H)$, we have another edge $g = (c_l^i(w), c_l^j(w))$ in $\mathsf{planar}(H)$ such that $\min(i,j) < \min(r,s) < \max(i,j) < \max(r,s)$. We pick g such that $\min(i,j)$ is minimum and break ties by picking one whose $\max(i,j)$ is maximum. Now, we have the following cases:

Case 1 ($i < r < j < s$): In this case, the edge $c_l^i(w), c_l^s(w)$ will be present in H. Since $i < p < s < q$, and $i < \min(r,s)$, this will contradict the way in which edge f was chosen.

Case 2 ($i < s < j < r$): In this case, the edge $(c_l^r(w), c_l^j(w))$ will be present in H. This edge will cross e and hence not be present in $\mathsf{planar}(H)$. Thus, we have an edge $g' = (c_l^{i'}(w), c_l^{j'}(w))$ in $\mathsf{planar}(H)$ such that $\min(i',j') < j < \max(i',j') < r$. We will thus have two subcases.

Case 2a ($i < \min(i',j')$): Here, we will have $i < \min(i',j') < j < \max(i',j')$. Hence this edge will cross g giving a contradiction to the first part of this lemma.

Case 2b ($\min(i',j') \leq i$): Here, this edge should have been chosen instead of g contradicting our choice of g.

The analysis of two remaining cases where $j < s < i < r$ and $j < r < i < s$ are similar to Cases 1 and 2 respectively. ◀

We next describe how to compute the triangulated planar graph $\widehat{H}$. Given H as an input, we first find $\mathsf{planar}(H)$ using Lemma 10. We then triangulate $\mathsf{planar}(H)$ by first adding edges in the boundary of each block as follows: let l be a block and v be a vertex in l. Let p be the smallest positive integer such that the vertex $c_l^p(v)$ is present in H. If the edge $(v, c_l^p(v))$ is not present in $\mathsf{planar}(H)$, we add this in $\widehat{H}$. This procedure does not result in a non-planar graph since no edge of $\mathsf{planar}(H)$ goes from one block to another. Every edge of l is now inside the boundary cycle. Finally, we triangulate the rest of the graph and add the triangulation edges to $\widehat{H}$. Note that this process can be done in logspace.

We will be using the following lemma which was proven by Imai et al.

▶ **Lemma 11** ([9]). *For every $\beta > 0$, there exists a polynomial time and $\tilde{O}(h^{1/2+\beta/2})$ space algorithm that takes a h-vertex planar graph P as input and outputs a set of vertices S, such that $|S|$ is $O(h^{1/2+\beta/2})$ and removal of S disconnects the graph into components of size $O(h^{1-\beta})$.*

For a subgraph H of $\mathsf{Aux}_\alpha(G)$, we construct the graph $\mathsf{psep}(H)$ in the following way.

(i) Construct $\widehat{H}$ from H.

(ii) Find a set S of vertices in $\widehat{H}$ which divides it into components of size $O(n^{1-\beta})$ by applying Lemma 11.

(iii) Add each vertex of S to the set $V(\mathsf{psep}(H))$ and each edge of $\widehat{H}(S)$ which is also in H to $E(\mathsf{psep}(H))$.

(iv) Let $e = (v, w)$ be a triangulation edge present in block l of $\widehat{H}$ whose both endpoints are in S. Let p and q be integers such that $w = c_l^p(v)$ and $v = c_l^q(w)$. We add the following set of at most four vertices and four edges to $\mathsf{psep}(H)$.

1. Let $p_1 < p$ be the largest integer such that an edge e_1 with endpoints v and $c_l^{p_1}(v)$ exists.
2. Let $p_2 > p$ be the smallest integer such that an edge e_2 with endpoints v and $c_l^{p_2}(v)$ exists.
3. Let $q_1 < q$ be the largest integer such that an edge e_3 with endpoints $c_l^{q_1}(w)$ and w exists.
4. Let $q_2 > q$ be the smallest integer such that an edge e_4 with endpoints $c_l^{q_2}(w)$ and w exists.

Note that the above edges could be directed either way. We add the vertices $c_l^{p_1}(v)$, $c_l^{p_2}(v)$, $c_l^{q_1}(w)$ and $c_l^{q_2}(w)$ to $V(\mathsf{psep}(H))$. For $i = 1, 2, 3, 4$, we add the edges e_i to $E(\mathsf{psep}(H))$. We call these four edges as *shadows* of e.

In Lemma 12 we show that the set $\mathsf{psep}(H)$ is a PseudoSeparator of H.

▶ **Lemma 12.** *Let G be a grid graph and H be a vertex induced subgraph of $\mathsf{Aux}_\alpha(G)$. The graph $\mathsf{psep}(H)$ is a $h^{1-\beta}$-PseudoSeparator of H.*

To prove Lemma 12, we first show a property of triangulated graphs that we use in our construction of PseudoSeparator. It is known that a simple cycle in a planar embedding of a planar graph divides the plane into two parts. We call these two parts the two *sides* of the cycle.

▶ **Lemma 13.** *Let G be a triangulated planar graph and S be a subset of its vertices. For every pair of vertex u, v which belong to different components of $G \setminus S$, there exists a cycle in $G[S]$, such that u and v belong to different sides of this cycle.*

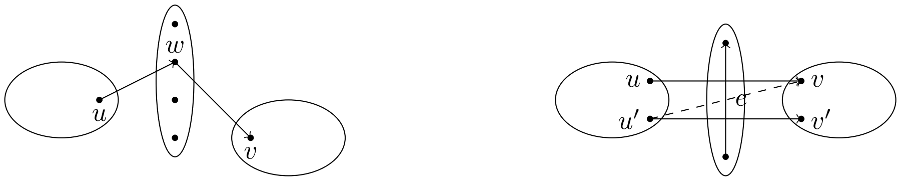
Proof. To prove the lemma, we first need some terminology. We call a set of faces an *edge-connected region* if it can be constructed in the following way:

- A set of a single face is an edge-connected region.
- If a set F is an edge-connected region and f is a face that shares an edge with one of the faces of F, then $F \cup \{f\}$ is an edge-connected region.

We can orient the edges of an undirected planar simple cycle to make it a directed cycle. This can help us identify the two *sides* of the cycle as *interior* (left-side) and *exterior* (right-side).

Let C be a component of $G \setminus S$ and S' be the set of vertices of S which are adjacent to at least one of the vertices of C in G. Let F be the set of triangle faces of G to which at least one vertex of C belongs. We first observe that for any face f of F, the vertices of f will either belong to C or S'. We see that F is a region of edge-connected faces. Miller proved that we could write the boundary of the region of edge-connected faces as a set of vertex-disjoint simple cycles with disjoint exteriors [11]. These cycles will contain only the vertices of S'. Hence the lemma follows. ◀

Proof of Lemma 12. Let $C = \mathsf{psep}(H)$. Let S be the set of vertices obtained from $\widehat{H}$ by using Lemma 11. We claim that if any two vertices u and v belong to different connected components of $\widehat{H} \setminus S$, then it belongs to diffent components of $\mathsf{cc}(H \diamond C)$. We prove this by contradiction. Let us assume that it is not true. Then there is an edge from e in H and two



(a) The $s-t$ path takes a vertex of the separator. **(b)** The $s-t$ path crosses an edge of the separator.

Figure 2

distinct sets U_1 and U_2 of $\widehat{H} \setminus S$ such that one of the end point of e is in U_1 and the other is in U_2. This edge e does not cross any of the edge of $\mathsf{psep}(H)$. Without loss of generality, let $e = (v, c_l^p(v))$, where $v \in U_1$ and $c_l^p(v)$ is not in U_1. (we pick the edge e such that p is minimum) Due to Lemma 13, it follows that there exists a triangulation edge f such that $f = ((c_l^q(v)), c_l^r(v))$ and that e crosses f. We orient the triangulation edge so that $q < p < r$. Now, since e is not present in $\mathsf{planar}(H)$, by Lemma 10 there exists at least one edge that crosses it and is present in $\mathsf{planar}(H)$. Let $g = (c_l^s(v), c_l^t(v))$ be one such edge such that $t - s$ is maximum. We thus have the following cases:

Case 1 ($s < q < p < r < t$): In this case, since g crosses e, by Lemma 6, we have that the edge $e' = (c_l^s(v), c_l^p(v))$ is also present in H. e' also crosses f. Since $p - s < p$, existence of e' contradicts our choice of e.

Case 2 ($q < t < p < s < r$): In this case, since g crosses e, by Lemma 6, we have that the edge $e' = (v, c_l^t(v))$ is also present in H. e' also crosses f. Since $t < p$, existence of e' contradicts our choice of e.

Case 3 ($q < s < p < t < r$): In this case, since g crosses e, by Lemma 6, we have that the edge $e' = (v, c_l^t(v))$ is present in H. e' also crosses f and hence e' was not present in $\mathsf{planar}(H)$. Thus there will exists an edge in $\mathsf{planar}(H)$ which crosses e' by Lemma 10. Let $g' = (c_l^{s'}(v), c_l^{t'}(v))$ be the edge in $\mathsf{planar}(H)$ that crosses e' such that $t' - s'$ is maximum. We see that $t' < q$ and $s' > r$, for otherwise, existence of g' will contradict the way g is chosen. Now, since the edges g and g' both cross e and g' is closer to v than g, by Lemma 7, the edge $e'' = (c_l^{s'}(v), c_l^t(v))$ will also be present in H. e'' will cross f. Now, any edge present in $\mathsf{planar}(H)$ that crosses e'' will contradict our choice of g or g'.

Case 4 ($t < q < p < r < s$): In this case, the edge $e' = (c_l^s(v), c_l^p(v))$ will also be present in H. e' will cross f and hence will not be present in $\mathsf{planar}(H)$. Any edge present in $\mathsf{planar}(H)$ that also crosses e' will contradict the way g is chosen.

In other cases, if g is picked such that one of its vertices is common with f, then e will cross a *shadow* edge of f giving a contradiction. If g is picked such that g cross f, then it contradicts the fact that both of them are present in $\widehat{H}$. ◀

Summarizing Lemmas 11 and 12 we have Theorem 14.

▶ **Theorem 14.** *Let G be a grid graph and H be a vertex induced subgraph of $\mathsf{Aux}_\alpha(G)$ with h vertices. For any constant $\beta > 0$, there exists an $\tilde{O}(h^{1/2+\beta/2})$ space and polynomial time algorithm that takes H as input and outputs an $h^{1-\beta}$*-PseudoSeparator *of size $O(h^{1/2+\beta/2})$.*

5 Algorithm to Solve Reachability in Auxiliary Graph

In this section, we discuss the grid graph reachability algorithm. Let G be a grid graph having $\widetilde{n}$ vertices. By induction we assume that we have access to a vertex induced subgraph H of $\mathsf{Aux}_\alpha(G)$, containing h vertices. Below we describe a recursive procedure $\mathsf{AuxReach}(H, x, y)$ that outputs true if there is a path from x to y in H and outputs false otherwise.

5.1 Description of the Algorithm AuxReach

First we construct a $h^{1-\beta}$-PseudoSeparator C of H, using Theorem 14. We also ensure that x and y are part of C (if not then we add them). Let $I_1, I_2, \ldots, I_l$ be the connected components of $H \diamond C$.

We maintain an array called visited of size $|C|$ to mark vertices or edges of the PseudoSeparator C. Each cell of visited corresponds to a distinct vertex or edge of C. For a vertex v in C, we set $\mathsf{visited}[v] := 1$ if there is a path from x to v in H, else it is set to 0. For an edge $e = (u, v)$ in C, we set $\mathsf{visited}[e] := u'$ if (i) there is an edge $f = (u', v')$ that crosses e, (ii) there is a path from x to u' in H and (iii) f is the closest such edge to u. Else $\mathsf{visited}[e]$ is set to NULL. Initially, for all vertex $v \in C$, $\mathsf{visited}[v] := 0$ and for all edges $e \in C$, $\mathsf{visited}[e] := \mathsf{NULL}$. We say that a vertex v is *marked* if either $\mathsf{visited}[v] := 1$ or $\mathsf{visited}[e] := v$ for some edge e.

First set $\mathsf{visited}[x] := 1$. We then perform an outer loop with h iteration and in each iteration update certain entries of the array visited as follows. For every vertex $v \in C$, the algorithm sets $\mathsf{visited}[v] := 1$ if there is a path from a marked vertex to v such that the internal vertices of that path all belong to only one component I_i. Similarly, for each edge $e = (u, v)$ of C, the algorithm sets $\mathsf{visited}[e] := u'$ if (i) there exists an edge $f = (u', v')$ which crosses e, (ii) there is a path from a marked vertex to u' such that the internal vertices of that path all belong to only one component I_i and, (iii) f is the closest such edge to u. Finally we output true if $\mathsf{visited}[y] = 1$ else output false. We use the procedure AuxReach recursively to check if there is a path between two vertices in a single connected component of $H \diamond C$. A formal description of AuxReach is given in Algorithm 1.

5.2 Proof of Correctness of AuxReach

Let P be a path from x to y in H. Suppose P passes through the components $I_{\sigma_1}, I_{\sigma_2}, \ldots, I_{\sigma_L}$ in this order. The length of this sequence can be at most $|H|$. As the path leaves the component I_{σ_j} and goes into $I_{\sigma_{j+1}}$, it can do in the following two ways only:

- **i.** The path exits I_{σ_j} through a vertex w of PseudoSeparator as shown in Figure 2a. In this case, Algorithm 1 would mark the vertex w.
- **ii.** The path exits I_{σ_j} through an edge (u, v) whose other endpoint is in $I_{\sigma_{j+1}}$. By Lemma 7, this edge will cross an edge $e = (x', y')$ of the PseudoSeparator. In this case, Algorithm 1 would mark the vertex u', such that there is an edge (u', v') that crosses e as well and (u', v') is closer than (u, v) to x' and there is a path in I_{σ_j} from a marked vertex to u'. By Lemma 7, the edge (u', v) would be present in H as well.

Thus after the j-th iteration, AuxReach would traverse the fragment of the path in the component I_{σ_j} and either mark its endpoint or a vertex which is closer to the edge e of C which the path crosses. Finally, t would be marked after L iterations if and only if there is a path from s to t in H. We give a formal proof of correctness in Lemma 15. For a path $P = (u_1, u_2, \ldots u_t)$, we define $\mathsf{tail}(P) := u_1$ and $\mathsf{head}(P) := u_t$.

▶ **Lemma 15.** *Let G be a grid graph and H be a vertex induced subgraph of $\mathsf{Aux}_\alpha(G)$. Then for any two vertices x, y in H, there is a path from x to y in H if and only if $\mathsf{AuxReach}(H, x, y)$ returns true.*

Proof. Firstly observe that a vertex is marked only if there is a path from some other marked vertex to that vertex in H. Hence if there is no path from x to y then y is never marked by AuxReach and hence AuxReach returns false.

Now let P be a path from x to y in H. We divide the path into subpaths $P_1, P_2, \ldots, P_l$, such that for each i, all vertices of P_i belong to $U \cup V(C)$ for some connected component U in $\mathsf{cc}(H \diamond C)$ and either (i) $\mathsf{head}(P_i) = \mathsf{tail}(P_{i+1})$, or (ii) $e_i = (\mathsf{head}(P_i), \mathsf{tail}(P_{i+1}))$ is an edge

Algorithm 1 AuxReach(H, s, t).

```
   Input: A vertex induced subgraph $H$ of $\mathsf{Aux}_\alpha(G)$ and two vertices $x$ and $y$ in $H$ (let
          $G$ be an $m \times m$ grid graph and $h = |V(H)|$)
   Output: true if there is a path from $x$ to $y$ in $H$ and false otherwise
 1 if $h \le m^{1/8}$ then Use DFS to solve the problem;   /* $m$ is a global variable
    where $G$ is an $m \times m$ grid graph */
 2 else
 3     Compute a $h^{1-\beta}$-PseudoSeparator $C$ of $H$ using Theorem 14;
 4     $C \leftarrow C \cup \{x, y\}$;
 5     foreach edge $e$ in $C$ do visited$[e] \leftarrow$ NULL;
 6     foreach vertex $v$ in $C$ do visited$[v] \leftarrow 0$;
 7     visited$[x] \leftarrow 1$;
 8     for $i = 1$ to $|H|$ do
 9         foreach edge $e = (u, v) \in C$ do
10             if $((\exists$ marked vertex $w) \cdot (\exists U \in \mathsf{cc}(H \diamond C)) \cdot (\exists f = (u', v')$ such that $f$
                crosses $e$ and $f$ is closest to $u) \cdot (\mathit{AuxReach}(H[U \cup \{w, u'\}], w, u') = \mathit{true}))$
                then
11                 visited$[e] \leftarrow u'$
12             end
13         end
14         foreach vertex $v \in C$ do
15             if $((\exists$ marked vertex $w) \cdot (\exists U \in$
                $\mathsf{cc}(H \diamond C)) \cdot (\mathit{AuxReach}(H[U \cup \{w, v\}], w, v) = \mathit{true}))$ then
16                 visited$[v] \leftarrow 1$
17             end
18         end
19     end
20     if $\mathit{visited}[y] = 1$ then return true;
21     else return false;
22 end
```

that crosses some edge $f_i \in C$. By Definition 8, we have that if condition (i) is true then $\mathsf{head}(P_i)$ is a vertex in C, and if condition (ii) is true then $\mathsf{head}(P_i)$ and $\mathsf{tail}(P_{i+1})$ belong to two different components of $\mathsf{cc}(H \diamond C)$ and e_i is the edge between them.

We claim that after i-th iteration of loop in Line 8 of Algorithm 1, either of the following two statements hold:

(I) $\mathsf{head}(P_i)$ is a vertex in C and $\mathsf{visited}[\mathsf{head}(P_i)] = 1$.

(II) There exists an edge $f_i = (u_i, v_i)$ of C such that the edge $e_i = (\mathsf{head}(P_i), \mathsf{tail}(P_{i+1}))$ crosses f_i and there is an edge $g_i = (u'_i, v'_i)$ which crosses f_i as well, such that g_i is closer to u_i than e_i and $\mathsf{visited}[f_i] = u'_i$.

We prove the claim by induction. The base case holds since x is marked at the beginning. We assume that the claim is true after the $(i-1)$-th iteration. We have that P_i belongs to $U \cup V(C)$ for some connected component U in $\mathsf{cc}(H \diamond C)$.

Case 1 **($\mathsf{head}(P_{i-1}) = \mathsf{tail}(P_i) = w$(say)):** By induction hypothesis w was marked after the $(i-1)$-th iteration. If $\mathsf{head}(P_i)$ is a vertex in C then it will be marked after the i-th iteration in Line 15. On the other hand if $e_i = (\mathsf{head}(P_i), \mathsf{tail}(P_{i+1}))$ is an edge that crosses some edge $f_i = (u_i, v_i) \in C$ then in the i-th iteration in Line 10, the algorithm marks a vertex u'_i such that, $g_i = (u'_i, v'_i)$ is the closest edge to u_i that crosses f_i and there is a path from w to u'_i.

Case 2 $\boldsymbol{(e_{i-1} = (\mathsf{head}(P_{i-1}), \mathsf{tail}(P_i))}$ **is an edge that crosses some edge** $\boldsymbol{f_{i-1} = (u_{i-1}, v_{i-1}) \in C}$**):** By induction hypothesis, there is an edge $g_{i-1} = (u'_{i-1}, v'_{i-1})$ which crosses f_{i-1} as well, such that g_{i-1} is closer to u_{i-1} than e_{i-1} and $\mathsf{visited}[f_{i-1}] = u'_{i-1}$. By Lemma 7 there is an edge in H between u'_{i-1} and $\mathsf{tail}(P_i)$ as well. Now if $\mathsf{head}(P_i)$ is a vertex in C then it will be marked after the i-th iteration in Line 15 by querying the graph $H[U \cup \{u'_{i-1}, \mathsf{head}(P_i)\}]$. On the other hand if $e_i = (\mathsf{head}(P_i), \mathsf{tail}(P_{i+1}))$ is an edge that crosses some edge $f_i = (u_i, v_i) \in C$ then in the i-th iteration in Line 10, AuxReach queries the graph $H[U \cup \{u'_{i-1}, u'_i\}]$ and marks a vertex u'_i such that, $g_i = (u'_i, v'_i)$ is the closest edge to u_i that crosses f_i and there is a path from u'_{i-1} to u'_i. ◀

Our subroutine would solve reachability in a subgraph H (having size h) of $\mathsf{Aux}_\alpha(G)$. We do not explicitly store a component of $\mathsf{cc}(H \diamond C)$, since it might be too large. Instead, we identify a component with the lowest indexed vertex present in it and use Reingold's algorithm on $H \diamond C$ to determine if a vertex is present in that component. We require $\widetilde{O}(h^{1/2+\beta/2})$ space to compute a $h^{1-\beta}$-PseudoSeparator by Theorem 14. We can potentially mark all the vertices of the PseudoSeparator and for each edge of PseudoSeparator we mark at most one additional vertex. Since the size of PseudoSeparator is at most $O(h^{1/2+\beta/2})$, we require $\widetilde{O}(h^{1/2+\beta/2})$ space. The algorithm recurses on a graph with $h^{1-\beta}$ vertices. Hence the depth of the recursion is at most $3/(\log(1-\beta)^{-1})$, which is a constant.

Since the graph H is given implicitly in our algorithm, there is an additional polynomial overhead involved in obtaining its vertices and edges. However, the total time complexity would remain a polynomial in the number of vertices since the recursion depth is constant.

▶ **Lemma 16.** *Let G be an $m \times m$ grid graph and H be a vertex induced subgraph of $\mathsf{Aux}_\alpha(G)$ with h vertices. For every $\beta > 0$, AuxReach runs in $\widetilde{O}(h^{1/2+\beta/2})$ space and polynomial time.*

Proof. Since the size of a component U in $\mathsf{cc}(H \diamond C)$ might be too large, we will not explicitly store it. Instead we identify a component by the lowest index vertex present in it and use Reingold's algorithm on $H \diamond C$ to determine if a vertex is present in U. Let $S(m, h)$ and $T(m, h)$ denote the space and time complexity functions respectively of AuxReach, where G is an $m \times m$ grid graph and h is the number of vertices in the graph H. As noted earlier the depth of the recursion is at most $d := 3/(\log(1-\beta)^{-1})$.

Consider $S(m, h)$ for any $h > m^{1/8}$. By Theorem 14, we require $\widetilde{O}(h^{1/2+\beta/2})$ space to execute Line 3. We can potentially mark all the vertex of C and for each edge e of C we store at most one additional vertex in $\mathsf{visited}[e]$. Since the size of C is at most $O(h^{1/2+\beta/2})$, we require $\widetilde{O}(h^{1/2+\beta/2})$ space to store C. By induction, a call to AuxReach in line 10 and 15 requires $S(m, h^{1-\beta})$ space which can be subsequently reused. Hence the space complexity satisfies the following recurrence. Then,

$$S(m,h) = \begin{cases} S(m, h^{1-\beta}) + \widetilde{O}(h^{1/2+\beta/2}) & h > m^{1/8} \\ \widetilde{O}(h) & \text{otherwise.} \end{cases}$$

Solving we get $S(m, h) = \widetilde{O}(h^{1/2+\beta/2} + m^{1/4})$.

Next we measure the time complexity of AuxReach. Consider the case when $h > m^{1/8}$. The total number of steps in AuxReach is some polynomial in h, say p. Moreover AuxReach makes q calls to AuxReach, where q is some other polynomial in h. Hence $q(h) \leq p(h)$. Then,

$$T(m,h) = \begin{cases} q \cdot T(h^{1-\beta}) + p & h > m^{1/8} \\ O(h) & \text{otherwise.} \end{cases}$$

Solving the above recurrence we get $T(m, h) = O(p \cdot q^d + m^{1/4}) = O(p^{2d} + m^{1/4})$. ◀

6 Solving Grid Graph

Let G be an $m \times m$ grid graph. As mentioned in the introduction, our objective is to run Algorithm 1 on the graph $\mathsf{Aux}_\alpha(G)$. By definition of $\mathsf{Aux}_\alpha(G)$, for every pair of vertices x, y in $\mathsf{Aux}_\alpha(G)$, there is a path from x to y in $\mathsf{Aux}_\alpha(G)$ if and only if there is a path from x to y in G. Hence it is sufficient to work with the graph $\mathsf{Aux}_\alpha(G)$. However, we do not have explicit access to the edges of $\mathsf{Aux}_\alpha(G)$. Note that we can obtain the edges of $\mathsf{Aux}_\alpha(G)$ by solving the corresponding subgrid of G to which that edge belongs. If the subgrid is small enough, then we use a standard linear space traversal algorithm. Otherwise, we use our algorithm recursively on the subgrid. Algorithm 2 outlines this method.

Algorithm 2 GridReach($\widehat{G}, \widehat{s}, \widehat{t}, m$).

```
Input: A grid graph $\widehat{G}$ and two vertices $\widehat{s}$, $\widehat{t}$ of $\widehat{G}$ and a positive integer $m$
Output: true if there is a path from $s$ to $t$ in $G$ and false otherwise
if $\widehat{G}$ is smaller than $m^{1/8} \times m^{1/8}$ grid then
    Use Depth-First Search to solve the problem;
end
else
    Use ImplicitAuxReach($\mathsf{Aux}_\alpha(G), \widehat{s}, \widehat{t}$) to solve the problem;
      /* ImplicitAuxReach executes the same way as AuxReach except for the
    case when it queries an edge $(u, v)$ in a block $B$ of $\mathsf{Aux}_\alpha(G)$.  In
    this case, the query is answered by calling GridReach($B, u, v, m$)
    where $B$ is the subgrid in which edge $(u, v)$ might belong.  */
end
```

Consider an $\widehat{m} \times \widehat{m}$ grid graph $\widehat{G}$. Let $S(\widehat{m})$ be the space complexity and $T(\widehat{m})$ be the time complexity of executing GridReach on $\widehat{G}$. Note that the size of $\mathsf{Aux}_\alpha(\widehat{G})$ is at most $\widehat{m}^{1+\alpha}$. For $\widehat{m} > m^{1/8}$, the space required to solve the grid-graph would be $S(\widehat{m}) = S(\widehat{m}^{1-\alpha}) + \widetilde{O}((\widehat{m}^{1+\alpha})^{1/2+\beta/2})$. This is because, a query whether $(u, v) \in \widehat{G}$ would invoke a recursion which would require $S(\widehat{m}^{1-\alpha})$ space and the main computation of ImplicitAuxReach could be done using $\widetilde{O}((\widehat{m}^{1+\alpha})^{1/2+\beta/2})$ space. Hence we get the following recurrence for space complexity.

$$S(\widehat{m}) = \begin{cases} S(\widehat{m}^{1-\alpha}) + \widetilde{O}((\widehat{m}^{1+\alpha})^{1/2+\beta/2}) & \widehat{m} > m^{1/8} \\ \widetilde{O}(\widehat{m}^{1/4}) & \text{otherwise} \end{cases}$$

Similar to the analysis of AuxReach, for appropriate polynomials p and q, the time complexity would satisfy the following recurrence:

$$T(\widehat{m}) = \begin{cases} q(\widehat{m}) \cdot T(\widehat{m}^{1-\alpha}) + p(\widehat{m}) & \widehat{m} > m^{1/8} \\ O(\widehat{m}) & \text{otherwise.} \end{cases}$$

Solving we get $S(m) = \widetilde{O}(m^{1/2+\beta/2+\alpha/2+\alpha\beta/2})$ and $T(m) = \mathsf{poly}(m)$. For any constant $\epsilon > 0$, we can chose α and β such that $S(m) = O(m^{1/2+\epsilon})$.

References

1 Eric Allender, David A Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. *Theory of Computing Systems*, 45(4):675–723, 2009.

2 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189(1):117–134, 2004. doi:10.1016/j.ic.2003.09.002.

3 Tetsuo Asano and Benjamin Doerr. Memory-Constrained Algorithms for Shortest Path Problem. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG 2011)*, 2011.

4 Tetsuo Asano, David Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe. $\tilde{O}(\sqrt{n})$-Space and Polynomial-Time Algorithm for Planar Directed Graph Reachability. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS 2014)*, pages 45–56, 2014.

5 Ryo Ashida and Kotaro Nakagawa. $\tilde{O}(n^{1/3})$-Space Algorithm for the Grid Graph Reachability Problem. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG 2018)*, pages 5:1–5:13, 2018.

6 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity. *SIAM Journal on Computing*, 27(5):1273–1282, 1998.

7 Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin F. Yang. New Time-Space Upperbounds for Directed Reachability in High-genus and H-minor-free Graphs. In *Proceedings of the 34th Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, pages 585–595, 2014.

8 Diptarka Chakraborty and Raghunath Tewari. An $O(n^{\epsilon})$ Space and Polynomial Time Algorithm for Reachability in Directed Layered Planar Graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(4):19:1–19:11, 2017.

9 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N. V. Vinodchandran, and Osamu Watanabe. An $O(n^{\frac{1}{2}+\epsilon})$-Space and Polynomial-Time Algorithm for Directed Planar Reachability. In *Proceedings of the 28th Conference on Computational Complexity (CCC 2013)*, pages 277–286, 2013.

10 Sampath Kannan, Sanjeev Khanna, and Sudeepa Roy. STCON in Directed Unique-Path Graphs. In *Proceedings of the 28th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*, volume 2, pages 256–267, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

11 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986.

12 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.

13 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

14 D. Stolee and N. V. Vinodchandran. Space-Efficient Algorithms for Reachability in Surface-Embedded Graphs. In *Proceedings of the 27th Annual Conference on Computational Complexity (CCC 2012)*, pages 326–333, 2012.

15 Avi Wigderson. The complexity of graph connectivity. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS 1992)*, pages 112–132. Springer, 1992.

Popular Roommates in Simply Exponential Time

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

Abstract

We consider the popular matching problem in a graph $G = (V, E)$ on n vertices with strict preferences. A matching M is *popular* if there is no matching N in G such that vertices that prefer N to M outnumber those that prefer M to N. It is known that it is NP-hard to decide if G has a popular matching or not. There is no faster algorithm known for this problem than the brute force algorithm that could take $n!$ time. Here we show a simply exponential time algorithm for this problem, i.e., one that runs in $O^*(k^n)$ time, where k is a constant.

We use the recent breakthrough result on the maximum number of stable matchings possible in such instances to analyze our algorithm for the popular matching problem. We identify a natural (also, hard) subclass of popular matchings called *truly popular* matchings and show an $O^*(2^n)$ time algorithm for the truly popular matching problem.

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

Keywords and phrases Roommates instance, Popular matching, Stable matching, Dual certificate

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.20

Acknowledgements Work done while visiting Max-Planck-Institut für Informatik, Saarland Informatics Campus, Germany. Thanks to Neeldhara Misra for asking me about fast exponential time algorithms for the popular roommates problem.

1 Introduction

Consider a matching problem in a graph $G = (V, E)$ on n vertices where each vertex has a strict ranking of its neighbors: such a graph is called a *roommates* instance. Matching M in G is *stable* if M has no blocking edge, i.e., an edge (u, v) such that both u and v prefer each other to their respective assignments in M. Stable matchings need not exist in G and a classical problem here is the *stable roommates* problem, i.e., does G admit a stable matching? There are several polynomial time algorithms [24, 30, 31] to solve this problem.

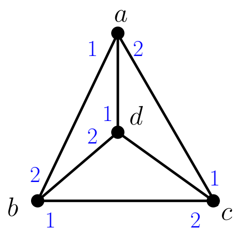
We consider a more relaxed notion of stability called *popularity*. A vertex u prefers matching M to matching N if either (i) u is matched in M and unmatched in N or (ii) u is matched in both M, N and prefers its partner in M to its partner in N. For any two matchings M_0 and M_1, let $\phi(M_0, M_1)$ be the number of vertices that prefer M_0 to M_1.

▶ **Definition 1.** *A matching M in $G = (V, E)$ is* popular *if $\phi(M, N) \geq \phi(N, M)$ for every matching N, i.e., $\Delta(N, M) \leq 0$ where $\Delta(N, M) = \phi(N, M) - \phi(M, N)$.*

In an election between M and N where vertices cast votes, $\phi(M, N)$ is the number of votes won by M and $\phi(N, M)$ is the number of votes won by N. By definition, a popular matching never loses an election to another matching; thus it is a weak *Condorcet winner* [5, 6] in the corresponding voting instance. Every stable matching in G is also popular [4, 17].

There are roommates instances with no stable matchings but with popular matchings, as shown in Fig. 1. Vertex a prefers b to c while b prefers c to a, and c prefers a to b. The last choice of a, b, c is d and d's preference is $a \succ b \succ c$. This instance has no stable matching, however it has 2 popular matchings $M_1 = \{(a, d), (b, c)\}$ and $M_2 = \{(a, c), (b, d)\}$.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 20; pp. 20:1–20:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



$a : b \succ c \succ d$

$b : c \succ a \succ d$

$c : a \succ b \succ d$

$d : a \succ b \succ c$

Figure 1 An instance with no stable matching, however it has two popular matchings. Numbers on edges indicate their preferences. The vertex d is the last choice of a, b, c and d's last choice is c.

Popular matchings need not always exist in a roommates instance. Consider the above instance without the vertex d. In any matching in the resulting instance, one of a, b, c (each is a top choice neighbor for some vertex) has to be left unmatched. Hence for any matching here, there is a more popular matching.

Popularity is a natural notion of "global stability" and popular matchings may exist in roommates instances with no stable matchings. The *popular roommates* problem is to decide if a given instance $G = (V, E)$ admits a popular matching or not. Unlike stable matchings, it is NP-hard to decide if a roommates instance admits a popular matching or not [13, 18]. There is no faster algorithm known for the popular roommates problem than the brute force algorithm that goes through all matchings in G and tests each for popularity. This algorithm could take $n!$ time. Can a faster algorithm be shown for the popular roommates problem?

1.1 Our results

Our main result is a simply exponential time algorithm for the popular roommates problem. Note that $O^*(k^n)$ denotes $O(k^n \cdot \mathsf{poly}(n))$.

▶ **Theorem 2.** *Given a roommates instance $G = (V, E)$ on n vertices with strict preferences, the popular roommates problem can be solved in $O^*(k^n)$ time, where k is a constant.*

When there is a cost function on the edge set, our algorithm also solves the min-cost popular matching problem. Regarding the constant k in the $O^*(k^n)$ running time, we show that $k \leq 3c$ where c is the constant involved in the recent breakthrough result [25] that showed an upper bound of c^n on the maximum number of stable matchings in a bipartite graph with n vertices on each side. It is known [25, 32] that $c_0 \leq c \leq 2^{17}$ where $c_0 \approx 2.28$.

We also identify a natural subclass of popular matchings called *truly popular* matchings; these are popular matchings that are also popular *fractional* matchings (defined in Section 2). The NP-hardness proof of the popular roommates problem [13] shows that the problem of deciding if a roommates instance admits a truly popular matching or not is NP-hard. We show an algorithm with running time $O^*(2^n)$ for the truly popular matching problem in a roommates instance G on n vertices.

▶ **Theorem 3.** *Given a roommates instance $G = (V, E)$ on n vertices with strict preferences, the problem of deciding whether G admits a truly popular matching or not can be solved in $O^*(2^n)$ time.*

1.2 Background and related results

The notion of popularity was proposed by Gärdenfors [17] in 1975 in bipartite graphs. Popular matchings always exist in bipartite graphs with strict preferences since stable matchings always exist here [16]. During the last 10-15 years, algorithms for popular matchings in bipartite graphs have been well-studied [1, 7, 8, 20, 22, 23, 26, 27, 28]: some of these results are in the domain of *one-sided* popularity, i.e., vertices on only one side of the bipartite graph have preferences.

In comparison, there are not many positive results for popular matchings in non-bipartite graphs. It was shown in [2] that given a matching M, it can be tested in polynomial time whether M is popular or not, even when there are ties in preference lists. It was shown in [21] that every roommates instance G admits a matching with *unpopularity factor* $O(\log n)$.

The popular roommates problem is NP-hard [13, 18]. In a complete graph on n vertices, this problem can be efficiently solved when n is odd, however it is NP-hard for even n [9]. The max-size popular matching problem is NP-hard even in instances with stable matchings (these are min-size popular matchings) [3]. The only known tractable subclasses of popular matchings are the class of stable matchings and the class of *strongly dominant* matchings [13] (a subclass of max-size popular matchings). When G has bounded treewidth, the min-cost popular matching problem can be solved in polynomial time [13].

There is a vast literature on fast exponential time algorithms for NP-hard problems and we refer to the book [15] on this subject. Fast exponential time algorithms for some hard problems in matchings under preferences are known: one such problem is the sex-equal stable marriage problem in bipartite graphs where the objective is to find a "fair" stable matching. When the length of preference lists of vertices on one side of the bipartite graph is bounded from above by a small value, a fast exponential time algorithm for finding a fair stable matching is known [29].

1.3 Our techniques

Let $G = (V, E)$ be the given roommates instance. It follows from LP-duality that every popular matching M in G has a *witness* to its popularity (Section 2 has these details). Witnesses have been used to show several results for popular matchings in bipartite graphs [13, 23, 27, 28]. Witnesses for popular matchings in non-bipartite graphs are more complicated than those in bipartite graphs. In non-bipartite graphs, witnesses have been used in [3, 9, 13] as *certificates* of popularity, i.e., to prove that certain matchings are popular.

In this paper we show a necessary condition for popularity in terms of witnesses. We then use this necessary condition to show a decomposition result for popular matchings: we show that every popular matching can be partitioned into a stable part and a *truly popular* part. Truly popular matchings are a new subclass of popular matchings introduced here and we characterize these matchings in terms of witnesses.

We use this characterization of truly popular matchings to show that every such matching can be realized as a stable matching in one of 2^n new roommates instances. In bipartite graphs, a mapping from a subset of max-size popular matchings to the set of stable matchings in a larger graph was shown in [8]. Our mapping from the set of truly popular matchings to the union of sets of stable matchings in 2^n graphs may be regarded as an extension of this. Our mapping is more complicated than the one in [8].

Organization of the paper. Section 2 discusses preliminaries. Witnesses for popular matchings and our main algorithmic result are in Section 3. Our fast exponential time algorithm for truly popular matchings is in Section 4.

2 Preliminaries

Our input is $G = (V, E)$ on n vertices and m edges where every vertex has a strict preference list ranking its neighbors. It would be convenient to regard every matching in G as a *perfect* matching, hence we augment G with self-loops so that every vertex is its own last choice neighbor. Thus any matching M in G becomes a perfect matching by including self-loops for vertices left unmatched.

Given a (perfect) matching M, consider the following edge weight function. For any edge (u, v) in E:

$$\text{let } \mathsf{wt}_M(u,v) \;=\; \begin{cases} 2 & \text{if } (u,v) \text{ is a blocking edge to } M \\ -2 & \text{if } u \text{ and } v \text{ prefer their respective partners in } M \text{ to each other} \\ 0 & \text{otherwise.} \end{cases}$$

For any edge (u, v), note that $\mathsf{wt}_M(u, v) = \mathsf{vote}_u(v, M(u)) + \mathsf{vote}_v(u, M(v))$, where for any pair of adjacent vertices u and v, $\mathsf{vote}_u(v, M(u))$ is u's vote for v versus $M(u)$: it is 1 if u prefers v to $M(u)$, it is -1 if u prefers $M(u)$ to v, and 0 otherwise, i.e., $v = M(u)$.

For any vertex u, define $\mathsf{wt}_M(u, u) = \mathsf{vote}_u(u, M(u))$ where $\mathsf{vote}_u(u, M(u)) = 0$ if the perfect matching M includes the self-loop (u, u), else $\mathsf{wt}_M(u, u) = -1$. For any perfect matching N, we have:

$$\mathsf{wt}_M(N) = \sum_{(u,v)\in N} \mathsf{wt}_M(u,v) = \sum_{u\in V} \mathsf{vote}_u(N(u), M(u)) = \phi(N,M) - \phi(M,N) = \Delta(N,M).$$

Matching M is popular if and only if $\Delta(N, M) = \mathsf{wt}_M(N) \le 0$ for all matchings N, i.e., if and only if every perfect matching in G with edge weight function wt_M has weight at most 0. Since $\mathsf{wt}_M(M) = 0$, a max-weight perfect matching has weight exactly 0. The max-weight perfect matching LP in G is described below.

$$\text{maximize} \sum_{e\in E'} \mathsf{wt}_M(e)\cdot x_e \qquad \text{(LP1)}$$

subject to

$$\textstyle\sum_{e\in\delta'(u)} x_e \;=\; 1 \quad \forall\, u \in V$$

$$\sum_{e\in E[B]} x_e \;\le\; \lfloor |B|/2 \rfloor \quad \forall B \in \Omega \qquad \text{and} \qquad x_e \;\ge\; 0 \quad \forall\, e \in E'.$$

Here E' is the set of edges in the graph G augmented with self-loops and $\delta'(u) = \delta(u) \cup \{(u, u)\}$ is the set of edges incident to u. Also, Ω is the collection of all odd-sized sets $B \subseteq V$ with $|B| \ge 3$. Note that $E[B]$ is the set of edges in E with both endpoints in B and self-loops do not belong to $E[B]$. Consider LP2: this is the dual LP corresponding to LP1.

$$\text{minimize} \sum_{u\in V} \alpha_u \;+\; \sum_{B\in\Omega} \lfloor\, |B|/2 \,\rfloor \cdot z_B \qquad \text{(LP2)}$$

subject to

$$\alpha_u + \alpha_v + \sum_{\substack{B\in\Omega\\ u,v\in B}} z_B \;\ge\; \mathsf{wt}_M(u,v) \quad \forall\, (u,v) \in E$$

$$\alpha_u \;\ge\; \mathsf{wt}_M(u,u) \quad \forall\, u \in V \qquad \text{and} \qquad z_B \;\ge\; 0 \quad \forall B \in \Omega.$$

Thus M is popular if and only if the optimal solution to LP2 is 0, i.e., if and only if there exists a feasible solution $(\vec{\alpha}, \vec{z})$ to LP2 such that $\sum_{u\in V} \alpha_u \;+\; \sum_{B\in\Omega} \lfloor\, |B|/2 \,\rfloor \cdot z_B = 0$.

▶ **Definition 4.** *For a popular matching M, an optimal solution $(\vec{\alpha}, \vec{z})$ to LP2 is called a* witness*.*

Popular fractional matchings. Recall that G is augmented with self-loops, so it has $m + n$ edges. A vector $\vec{p} \in \mathbb{R}^{m+n}_{\geq 0}$ such that $\sum_{e \in \delta'(u)} p_e = 1$ for all vertices u is a (perfect) fractional matching in G. The notion of popularity extends to fractional matchings as well. Here we compare an integral matching M with a fractional matching $\vec{p}$ as follows:

$$\Delta(\vec{p}, M) \ = \ \sum_{u \in V} \mathsf{vote}_u(\vec{p}, M) \ = \ \sum_{u \in V} \sum_{v \in \mathsf{Nbr}'(u)} p_{(u,v)} \cdot \mathsf{vote}_u(v, M(u)),$$

where $\mathsf{Nbr}'(u) = \mathsf{Nbr}(u) \cup \{u\}$. Note that $\mathsf{Nbr}(u)$ is the set of u's neighbors in the original graph G (without self-loops).

An integral matching M is a *popular fractional* matching if $\Delta(\vec{p}, M) \leq 0$ for all fractional matchings $\vec{p}$ in G. Every popular matching in G need not be a popular fractional matching. See the instance G in Fig. 2 where vertex preferences are indicated on edges.

Here a is the top choice of b, c, s while b and c are each other's second choices. Vertex a's preference order is $b \succ c \succ s$. Vertex q's order is $r \succ s$ and r's order is $s \succ q$, and s's order is $a \succ q \succ r$.

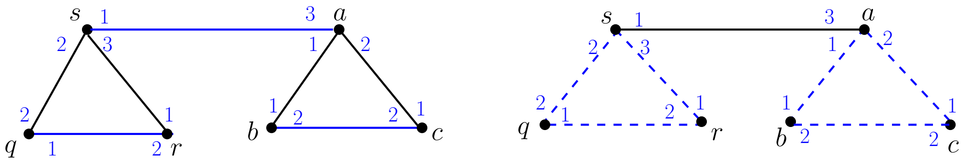


Figure 2 The half-integral matching on the right with a value of 1/2 on the dashed edges is more popular than $M = \{(a, s), (b, c), (q, r)\}$. Note that M is a popular matching.

▷ Claim 5. $M = \{(a, s), (b, c), (q, r)\}$ is popular in G (see Fig. 2), however M is not a popular fractional matching in G.

Proof. We prove the popularity of M via the witness $(\vec{\alpha}, \vec{z})$ where $\alpha_a = \alpha_r = 1$ and $\alpha_b = \alpha_c = \alpha_q = \alpha_s = -1$ along with $z_{\{a,b,c\}} = 2$ and $z_B = 0$ for all other odd sets B. It is easy to check that $(\vec{\alpha}, \vec{z})$ satisfies the constraints in LP2. Also $\sum_u \alpha_u \ + \ \sum_B \lfloor |B|/2 \rfloor z_B = 2 - 4 + 2 = 0$. Thus M is popular in G.

However M is not a popular fractional matching in G. We will show a more popular fractional matching. Consider the half-integral matching $\vec{p}$ indicated on the right in Fig. 2. So $p_e = 1/2$ for $e \in \{(a, b), (b, c), (c, a), (q, r), (r, s), (s, q)\}$. We have $\Delta(\vec{p}, M) = 5/2 - 3/2 = 1$ since $\vec{p}$ gets the vote of a and 1/2-votes of b, c, r while M gets the vote of s and 1/2-vote of q. Thus $\vec{p}$ defeats M, so M is not a popular fractional matching in G. ◁

Hence a popular matching may lose an election against a fractional matching. We introduce the following natural subclass of popular matchings.

▶ **Definition 6.** *A matching M in G is* truly popular *if M is a popular fractional matching.*

Thus M is a truly popular matching if $\Delta(\vec{p}, M) \leq 0$ for all fractional matchings $\vec{p}$. The NP-hardness proof of popular roommates problem in [13] implies that the problem of deciding if a roommates instance G admits a truly popular matching or not is also NP-hard.

Note that a roommates instance may admit popular matchings but *no* truly popular matching. For instance, $M = \{(a, s), (b, c), (q, r)\}$ is the only popular matching in the instance given in Fig. 2 and we know from Claim 5 that M is not truly popular.

3 An algorithm for the popular roommates problem

In this section we show that every popular matching admits a witness with certain structure. This will be used in a structural decomposition result and our algorithm for the popular roommates problem is based on this decomposition.

3.1 Popular matchings and witnesses

In this section we study witnesses for popular matchings. Our first result is the following.

▶ **Lemma 7.** *Let M be a popular matching in G. Then M has a witness $(\vec{\alpha}, \vec{z})$ such that $\vec{\alpha} \in \{0, \pm 1\}^n$ and $z_B \in \{0, 1, 2\}$ for all $B \in \Omega$.*

Proof. Let M be a popular matching in G. Consider LP1 from Section 2: this is the LP for max-weight perfect matching in the graph G augmented with self-loops and with edge weights given by wt_M. Since $\mathsf{wt}_M(M) = \Delta(M, M) = 0$, the characteristic vector of M is an optimal solution to LP1. The constraint system corresponding to LP1 is totally dual integral (TDI) [10]. Thus there is an optimal integral solution $(\vec{\alpha}, \vec{z})$ to the dual LP, i.e., LP2.

We have $\alpha_u \geq \mathsf{wt}_M(u, u) \geq -1$ for all vertices u. Moreover, if $(u, u) \in M$, this constraint is tight by complementary slackness: so $\alpha_u = \mathsf{wt}_M(u, u) = 0$ for such a vertex u. Similarly, for a vertex u matched to a non-trivial neighbor in M (say, $(u, v) \in M$), we have by complementary slackness:

$$\alpha_u + \alpha_v + \sum_{\substack{B \in \Omega \\ u,v \in B}} z_B \;=\; \mathsf{wt}_M(u, v) = 0. \tag{1}$$

Since $z_B \geq 0$ for all B, this means $\alpha_u + \alpha_v \leq 0$, so $\alpha_u \leq -\alpha_v \leq 1$. Hence $\vec{\alpha} \in \{0, \pm 1\}^n$. Let $B \in \Omega$ be such that $z_B > 0$. Then complementary slackness on LP1 implies:

$$\sum_{e \in E[B]} x_e = \lfloor |B|/2 \rfloor. \tag{2}$$

Since $|B| \geq 3$, any $B \in \Omega$ with $z_B > 0$ has at least 1 matched edge in it. Let $(u, v) \in M \cap E[B]$. Then non-negativity of z_B-values and (1) imply that $z_B \leq -(\alpha_u + \alpha_v) \leq 2$. Hence $z_B \in \{0, 1, 2\}$ for every $B \in \Omega$. ◀

We now characterize truly popular matchings in terms of witnesses.

▶ **Theorem 8.** *A matching M is truly popular iff M has a witness $(\vec{\alpha}, \vec{z})$ such that $\vec{\alpha} \in \{0, \pm 1\}^n$ and $\vec{z} = \vec{0}$.*

Proof. We assume G is augmented with self-loops, so any fractional matching $\vec{p}$ becomes a perfect fractional matching by using self-loops. For any perfect fractional matching $\vec{p}$ in G: (recall that $E' = E \cup \{(u, u) : u \in V\}$)

$$\mathsf{wt}_M(\vec{p}) \;=\; \sum_{e \in E'} p_e \cdot \mathsf{wt}_M(e) \;=\; \sum_{u \in V} \sum_{v \in \mathsf{Nbr}'(u)} p_{(u,v)} \cdot \mathsf{vote}_u(v, M(u)) \;=\; \Delta(\vec{p}, M).$$

Thus M is a popular fractional matching if and only if $\mathsf{wt}_M(\vec{p}) = \Delta(\vec{p}, M) \leq 0$ for all fractional matchings $\vec{p}$. Consider LP3 given below. LP3 is the max-weight perfect fractional matching LP in the graph G with edge weight function wt_M. LP4 is the dual of LP3.

Suppose M is a matching in G with a witness $(\vec{\alpha}, \vec{0})$ for some $\vec{\alpha} \in \{0, \pm 1\}^n$. So:
(i) $\sum_u \alpha_u = 0$, (ii) $\alpha_v \geq \mathsf{wt}_M(v, v)\ \forall\, v \in V$, and (iii) $\alpha_u + \alpha_v \geq \mathsf{wt}_M(u, v)\ \forall\, (u, v) \in E$.

$$\max \sum_{e \in E'} \mathsf{wt}_M(e) \cdot x_e \qquad \text{(LP3)}$$

$$\text{s.t.} \quad \sum_{e \in \delta'(u)} x_e = 1 \qquad \forall\, u \in V$$

$$x_e \geq 0 \qquad \forall\, e \in E'.$$

$$\min \sum_{u \in V} \alpha_u \qquad \text{(LP4)}$$

$$\text{s.t.} \quad \alpha_u + \alpha_v \geq \mathsf{wt}_M(u,v) \qquad \forall\, (u,v) \in E$$

$$\alpha_u \geq \mathsf{wt}_M(u,u) \qquad \forall\, u \in V$$

It follows from properties (ii) and (iii) stated above that $\vec{\alpha}$ is a feasible solution to LP4. It follows from property (i) that the optimal value of LP4 is at most 0. Thus the optimal value of LP3 is at most 0. Since $\mathsf{wt}_M(M) = \Delta(M, M) = 0$, this means that M is an optimal solution to LP3. So $\mathsf{wt}_M(\vec{p}) \leq \mathsf{wt}_M(M) = 0$ for all fractional matchings $\vec{p}$. Thus $\Delta(\vec{p}, M) \leq 0$ for all fractional matchings $\vec{p}$, i.e., M is a popular fractional matching.

Conversely, suppose M is a truly popular matching in G. So M is a popular fractional matching in G. Hence $\Delta(\vec{p}, M) \leq 0$ for all perfect fractional matchings $\vec{p}$, thus $\mathsf{wt}_M(\vec{p}) = \Delta(\vec{p}, M) \leq 0$. Since $\mathsf{wt}_M(M) = 0$, this means M is an optimal solution to LP3.

▷ Claim 9. LP4 has an optimal solution that is integral.

The proof of Claim 9 is given below. Let $\vec{\alpha}$ be an optimal solution of LP4 that is integral. We have $\alpha_u \geq \mathsf{wt}_M(u,u)$ from the constraints. Since $\mathsf{wt}_M(u,u) \geq -1$, we have $\alpha_u \geq -1$ for all vertices u. It follows from complementary slackness conditions that $\alpha_u + \alpha_v = \mathsf{wt}_M(u,v) = 0$ for every edge $(u,v) \in M$. Since $\alpha_v \geq -1$, it follows that $\alpha_u \leq 1$.

It also follows from complementary slackness conditions that $\alpha_u = \mathsf{wt}_M(u,u) = 0$ for every vertex u matched in M along the self-loop (u,u). Thus M has a witness $(\vec{\alpha}, \vec{0})$ such that $\vec{\alpha} \in \{0, \pm 1\}^n$. ◀

Proof of Claim 9. Let $\vec{\alpha}$ be any extreme point of the feasible region of LP4. So we have $A\vec{\alpha} = b$ for some submatrix A of the constraint matrix of LP4. Some of the tight constraints are of the type $\alpha_u = \mathsf{wt}_M(u,u)$: this immediately implies α_u is either 0 or -1, i.e., these coordinates in $\vec{\alpha}$ are integral. Let us remove these constraints from $A\vec{\alpha} = b$, so we have $A'\vec{\alpha}' = b'$ where all the constraints are of the type $\alpha_u + \alpha_v = \mathsf{wt}_M(u,v)$ for $(u,v) \in E$. So $\vec{\alpha}' = A'^{-1} \cdot b'$.

It is easy to see that all entries in A'^{-1} are half-integral. This follows from the fact that the fractional matching polytope of G is half-integral: this is due to the integrality of the fractional matching polytope in bipartite graphs (Birkhoff-von Neumann theorem).

Since $\mathsf{wt}_M(e) \in \{0, \pm 2\}$ for every $e \in E$, every entry in b' is an even integer. Hence $\vec{\alpha}' = A'^{-1} \cdot b'$ is an integral vector. Thus $\vec{\alpha}$ is integral. ◁

Hence M is a truly popular matching if and only if M has a witness $(\vec{\alpha}, \vec{0})$ such that $\vec{\alpha} \in \{0, \pm 1\}^n$. For the sake of brevity, we will say $\vec{\alpha}$ is a witness of M.

3.2 A decomposition result for popular matchings

The following theorem shows that every popular matching in G can be partitioned into a stable part and a *truly popular* part. This decomposition resembles a result from [8] that shows that every popular matching M in a bipartite graph can be decomposed into a stable part and a *dominant*[1] part.

[1] A popular matching N is dominant if N is more popular than any larger matching.

▶ **Theorem 10.** *Let M be a popular matching in $G = (V, E)$. Then $M = M_0 \cup M_1$ such that*

1. *M_0 is stable in the subgraph induced on some subset $C \subseteq V$;*
2. *M_1 is truly popular in the subgraph induced on $V \setminus C$.*

Proof. We know from Lemma 7 that every integral witness $(\vec{\alpha}, \vec{z})$ of a popular matching M satisfies $\vec{\alpha} \in \{0, \pm 1\}^n$ and $z_B \in \{0, 1, 2\}$ for all $B \in \Omega$. Let $(\vec{\alpha}, \vec{z})$ be an integral witness of M such that the sets B with $z_B > 0$ form a laminar family $\mathcal{B}$. The primal-dual algorithm of Edmonds [12] shows that M has such a witness.

Let $B_1, \ldots, B_k$ be the maximal sets in $\mathcal{B}$. We know from (2) that each $B_i \in \mathcal{B}$ has $\lfloor |B_i|/2 \rfloor$ edges of M within it. For $1 \le i \le k$, let b_i be the lone vertex in B_i that is *not* matched to a vertex inside B_i. That is, every vertex in $B_i \setminus \{b_i\}$ is matched in M to another vertex in $B_i \setminus \{b_i\}$. Let C denote the vertex set $\cup_{i=1}^{k} (B_i \setminus \{b_i\})$.

Let M_0 be the matching M restricted to the subgraph induced on C and let M_1 be the matching M restricted to the subgraph induced on $V \setminus C$. Observe that $M = M_0 \cup M_1$. Claim 11 and Claim 12 show that M_0 and M_1 are what we seek.

▷ Claim 11. The matching M_0 is stable in the subgraph induced on C.

▷ Claim 12. The matching M_1 is truly popular in the subgraph induced on $V \setminus C$.

Claim 11 and Claim 12 are proved below. This finishes the proof of Theorem 10. ◀

Proof of Claim 11. We will prove the stability of M_0 by showing that no edge with both endpoints in C blocks M. Consider any edge (u, v) with $u, v \in C$. We know that $\alpha_u + \alpha_v + \sum_{\substack{B \in \mathcal{B} \\ u,v \in B}} z_B \ge \mathsf{wt}_M(u, v)$.

$\mathcal{B}$ is a laminar family. Let $\mathcal{B}' \subseteq \mathcal{B}$ be the collection of sets with both u and v. We need to bound $\sum_{B \in \mathcal{B}'} z_B$. Let B' be the minimal set in $\mathcal{B}'$. It follows from (2) that the partner of at least one of u, v (say, u) is in B' and hence in every set in $\mathcal{B}'$. So we can use (1) for the pair $u, M(u)$ to bound $\sum_{B \in \mathcal{B}'} z_B$. Since $\alpha_u, \alpha_{M(u)} \ge -1$, we have $\sum_{B \in \mathcal{B}'} z_B \le 2$.

The definition of C implies that every vertex $x \in C$ is matched in M to another vertex $M(x)$ in C. Moreover there is some $B_i \in \mathcal{B}$ such that $x, M(x) \in B_i$. Thus $\sum_{B \in \mathcal{B}: x, M(x) \in B} z_B$ is at least 1 and so $\alpha_x + \alpha_{M(x)} \le -1$ by (1). Hence α_x is in $\{0, -1\}$ for every $x \in C$.

Suppose $\alpha_u = 0$. Then $\alpha_u + \alpha_{M(u)} + \sum_{B: u, M(u) \in B} z_B = \mathsf{wt}_M(u, M(u)) = 0$ along with $\alpha_u = 0$ and $\alpha_{M(u)} \ge -1$ implies that $\sum_{B: u, M(u) \in B} z_B \le 1$. Since this sum is integral and positive, it equals 1. So $\mathsf{wt}_M(u, v) \le 1$ in this case. Similarly, when $\alpha_u = -1$, $\mathsf{wt}_M(u, v) \le \alpha_u + \alpha_v + \sum_{B \in \mathcal{B}'} z_B \le -1 + 0 + 2 = 1$. Hence in both cases, $\mathsf{wt}_M(u, v) \le 0$ (since it is in $\{0, \pm 2\}$).

So there is no blocking edge to M with both endpoints in C. Thus M_0 is stable in the subgraph induced on C. ◁

Proof of Claim 12. Let $(\vec{\alpha}, \vec{z})$ be M's witness using which C was defined. We claim $(\vec{\alpha}, \vec{0})$ is a witness for M_1 in the subgraph induced on $V \setminus C$. So we need to show that $\sum_{u \in V \setminus C} \alpha_u = 0$ and $\alpha_u + \alpha_v \ge \mathsf{wt}_{M_1}(u, v) = \mathsf{wt}_M(u, v)$ for every edge (u, v) in this subgraph. We already know that $\alpha_u \ge \mathsf{wt}_{M_1}(u, u) = \mathsf{wt}_M(u, u)$ for all $u \in V$.

- We have $\alpha_u + \alpha_v + \sum_{B: u,v \in B} z_B \ge \mathsf{wt}_M(u, v)$ for every edge (u, v) in G. There is no $B \in \mathcal{B}$ that contains two vertices in $V \setminus C$. Thus $\sum_{B: u,v \in B} z_B = 0$ and so we have the desired constraint $\alpha_u + \alpha_v \ge \mathsf{wt}_M(u, v)$ for every edge (u, v) in this subgraph.
- For any vertex $u \in V \setminus C$ that is matched in M, its partner $M(u) = v$ is also in $V \setminus C$ and we have $\alpha_u + \alpha_v = \mathsf{wt}_M(u, v) = 0$ by complementary slackness (see (1)). For any vertex u matched in M along its self-loop, $\alpha_u = \mathsf{wt}_M(u, u) = 0$. Thus $\sum_{u \in V \setminus C} \alpha_u = 0$. ◁

The proof of Theorem 10 allows us to show a more structured partition of popular matchings as stated in Lemma 13 below. Call a truly popular matching M *special* if M admits a witness $\vec{\alpha} \in \{\pm 1\}^n$.

▶ **Lemma 13.** *Let M be a popular matching in $G = (V, E)$. Then $M = M'_0 \cup M'_1$ where M'_0 is a stable matching in the subgraph induced on some $U \subseteq V$ and M'_1 is a special truly popular matching in the subgraph induced on $V \setminus U$.*

Proof. We will use Theorem 10 here. Let $\mathcal{B} \subseteq \Omega$, $C \subseteq V$, and $\vec{\alpha} \in \{0, \pm 1\}^n$ be as defined in the proof of Theorem 10. Let $U = C \cup \{u \in V \setminus C : \alpha_u = 0\}$.

Let M'_0 be the matching M restricted to the subgraph induced on U. Since $U \supseteq C$, we have $M'_0 \supseteq M_0$, where M_0 was defined in Theorem 10. We claim M'_0 is stable in the subgraph induced on U. It follows from the proofs of Claim 11 and Claim 12 that there is no blocking edge (u, v) to M'_0 where both $u, v \in C$ or both $u, v \in U \setminus C$ (in this case $\alpha_u = \alpha_v = 0$). So what we need to show now is that there is no blocking edge (u, v) to M'_0 where $u \in C$ and $v \in U \setminus C$.

If there is no $B \in \mathcal{B}$ such that $u, v \in B$ then $\mathsf{wt}_M(u, v) \le \alpha_u + \alpha_v \le 0$. Suppose there is some $B \in \mathcal{B}$ with $u, v \in B$. It follows from (2) and the definition of C that u and its partner $M(u)$ are in B. We know from the proof of Claim 11 that either (i) $\alpha_u = -1$ or (ii) $\alpha_u = 0$ and $\sum_{B:u,M(u)\in B} z_B \le 1$. Since $\alpha_v = 0$, this means that $\alpha_u + \alpha_v + \sum_{B:u,v\in B} z_B \le 1$. So $\mathsf{wt}_M(u, v) \le 1$, i.e., $\mathsf{wt}_M(u, v) \le 0$ (since it is even). Thus (u, v) does *not* block M.

So M'_0 is stable in the subgraph induced on U. Let M'_1 be the matching M restricted to the subgraph induced on $V \setminus U$. It follows from the definition of U that M'_1 has a witness $\vec{\alpha}$ where $\alpha_u \in \{\pm 1\}$ for all $u \in V \setminus U$. Hence M'_1 is a special truly popular matching in the subgraph induced on $V \setminus U$. ◀

3.3 Our algorithm

We present our algorithm for the popular roommates problem. The input is $G = (V, E)$.

```
1. For each $U \subseteq V$ do:
   a. For each stable matching $S$ in the subgraph induced on $U$ do:
   b. For each special truly popular matching $T$ in the subgraph induced on $V \setminus U$ do:
      - If $S \cup T$ is popular in $G$ then return $S \cup T$.
2. Return "$G$ has no popular matching".
```

A matching M can be tested for popularity via LP1 (see Section 2). There are also combinatorial algorithms [2, 22] to check if a given matching in a roommates instance is popular or not. Lemma 13 shows that every popular matching M admits a decomposition as $M = S \cup T$ where S is stable in some subgraph and T is a special truly popular matching in the remaining part of G. Thus if no matching of the form $S \cup T$ is popular then G has no popular matching. This proves the correctness of our algorithm.

Implementation. All stable matchings in the graph $G_U = (U, E')$ induced on U can be listed by enumerating all stable matchings in the bipartite graph $G'_U = (U' \cup U'', E'')$ [11] where $U' = \{u' : u \in U\}$ and $U'' = \{u'' : u \in U\}$; for every edge (u, v) in G_U, there are 2 edges (u', v'') and (v', u'') in G'_U. Preferences in G'_U are inherited from G_U. Every matching in the bipartite graph G'_U becomes a half-integral matching in the given graph G_U.

It is known how to enumerate all stable matchings in a bipartite graph in $O^*(s)$ time where s is the number of stable matchings in this bipartite graph [19]. It was recently shown [25] that the maximum number of stable matchings possible in a bipartite graph with n vertices on each side is c^n for some constant c. Thus in $O^*(c^n)$ time we can enumerate all stable matchings in a roommates instance on n vertices.

We bound the running time of our algorithm via the following bound on the number of "special truly popular" matchings present in a roommates instance. Here c is the constant from [25] that was used in the paragraph above.

▶ **Lemma 14.** *A roommates instance H on t vertices has at most $(2c)^t$ special truly popular matchings.*

The proof of Lemma 14 shows that every special truly popular matching in H can be realized as a stable matching in one of 2^t roommates instances, each on t vertices. This proof is given in Section 4.1.

Running time of our algorithm. The total number of candidate matchings tested by our algorithm is at most:

$$\sum_{i=0}^{n}\binom{n}{i}\cdot c^i\cdot(2c)^{n-i} \quad = \quad c^n\cdot\sum_{i=0}^{n}\binom{n}{i}2^{n-i} \quad = \quad (3c)^n.$$

In the summation above, c^i is the bound on the number of stable matchings in the subgraph G_U induced on U (where $|U| = i$) and the second term, which is $(2c)^{n-i}$, is the bound on the number of special truly popular matchings in the subgraph G_W induced on $W = V \setminus U$ (note that $|V \setminus U| = n - i$). This proves Theorem 2 stated in Section 1.

4 Truly popular matchings

In this section we use the characterization of truly popular matchings from Theorem 8 to show a fast exponential time algorithm for the problem of deciding if G admits a truly popular matching or not. Our algorithm goes through all $S \subseteq V$ and checks if there is a popular matching in G with a witness $\vec{\alpha}$ such that $\alpha_v = 0$ for all $v \in S$ and $\alpha_v \in \{\pm 1\}$ for all $v \in V \setminus S$. So the problem we look to efficiently solve is:

* given $S \subseteq V$, is there a truly popular matching in G with a witness $\vec{\alpha} \in \{0, \pm 1\}^n$ such that $\alpha_v = 0$ if and only if $v \in S$.

We will now show an efficient algorithm for the above problem. We solve this problem by posing it as a stable roommates problem with forbidden edges, which can be solved in linear time [14]. Given any subset $S \subseteq V$, we will construct a new roommates instance $G_S = (V_S, E_S)$ as follows. The vertex set $V_S = \{u_0 : u \in S\} \cup \{u_-, u_+, \ell(u) : u \in V \setminus S\}$.

The vertex $\ell(u)$ will be called a *dummy vertex* as its purpose is to ensure that only *one* of u_+, u_- can be matched to a non-dummy neighbor, i.e., an element in $\{v_+, v_0, v_- : v \in \mathsf{Nbr}(u)\}$. The edge set E_S consists of the following edges:

- For every $(u, v) \in E$ where $u, v \in S$: the edge $(u_0, v_0) \in E_S$.
- For every $(u, v) \in E$ where $u \in V \setminus S$ and $v \in S$: the edge $(u_+, v_0) \in E_S$.
- For every $(u, v) \in E$ where $u, v \in V \setminus S$: if u prefers v to every neighbor in S then $(u_-, v_+) \in E_S$.

Also, for every vertex $u \in V \setminus S$: the edges $(u_+, \ell(u))$ and $(u_-, \ell(u))$ are in E_S. The preference order of vertices in V_S is as follows.

1. For any dummy vertex $\ell(u)$: the order is $u_+ \succ u_-$.
2. For any subscript 0 vertex u_0: the order among its neighbors is as per u's original preference order in G. Suppose u's preference order in G is: $a \succ b \succ c \succ d$ where $a, c \in S$ and $b, d \in V \setminus S$, then u_0's neighbors in G_S are a_0, b_+, c_0, d_+ and u_0's preference order is: $a_0 \succ b_+ \succ c_0 \succ d_+$.

3. For any subscript $+$ vertex u_+: the order among its neighbors in G_S is as per u's preference order in G with $\ell(u)$ as its least preferred vertex.
4. For any subscript $-$ vertex u_-: the order among its neighbors is $\ell(u)$ as its top choice followed by its other neighbors in G_S as per u's preference order in G.

The following theorem shows the equivalence we need.

▶ **Theorem 15.** *The instance G admits a truly popular matching with a witness $\vec{\alpha}$ where $\alpha_u = 0$ for $u \in S$ and $\alpha_v \in \{\pm 1\}$ for $v \in V \setminus S$ iff G_S has a stable matching M_S with the following properties:*

1. *M_S avoids all edges between a subscript 0 vertex and a subscript $+$ vertex;*
2. *M_S matches all subscript $-$ vertices.*

Proof. Suppose G admits a truly popular matching T_S with such a witness $\vec{\alpha}$. We will show a desired stable matching M_S in G_S. For any vertex u, let $s_u = +/-/0$ corresponding to $\alpha_u = +1/-1/0$, respectively. For any vertex $u \in V \setminus S$, we have $\alpha_u \in \{\pm 1\}$ and so $s_u \in \{\pm\}$; if $s_u = +$ then let $t_u = -$, else let $t_u = +$.

Let $M_S = \{(u_{s_u}, v_{s_v}) : (u, v) \in T_S\} \cup \{(u_{t_u}, \ell(u)) : u \in V \setminus S\}$.

▷ Claim 16. $M_S \subseteq E_S$, i.e., for every (u, v) in T_S, the edge (u_{s_u}, v_{s_v}) is present in G_S.

Proof. Since T_S is truly popular, the characteristic vector of T_S is an optimal solution of LP3. We also know that $\vec{\alpha}$ is an optimal solution of LP4. It follows from complementary slackness conditions on LP3 and LP4 that for every edge $(u, v) \in T_S$, $\alpha_u + \alpha_v = \mathsf{wt}_{T_S}(u, v)$. Since $\mathsf{wt}_{T_S}(u, v) = 0$ for any edge $(u, v) \in T_S$, either $\alpha_u = \alpha_v = 0$ or $\{\alpha_u, \alpha_v\} = \{-1, 1\}$. So every edge in M_S that is not incident to any ℓ-vertex is of the type either (u_0, v_0) or (u_+, v_-).

For every edge (u, v) in G where $\alpha_u = \alpha_v = 0$, the edge (u_0, v_0) is in G_S. Consider an edge (u, v) in T_S where $\alpha_u = -1$. We need to show that (u_-, v_+) is in G_S. Since $\vec{\alpha}$ is a witness of T_S, we have $\mathsf{wt}_{T_S}(u, r) \le \alpha_u + \alpha_r = -1 + 0 = -1$ for every neighbor $r \in S$. Since $\mathsf{wt}_{T_S}(e) \in \{0, \pm 2\}$ for all $e \in E$, this means $\mathsf{wt}_{T_S}(u, r) = -2$, i.e., u prefers its partner in T_S (this is v) to r. Since this constraint holds for every $r \in S \cap \mathsf{Nbr}(u)$, it follows from the definition of E_S that $(u_-, v_+) \in E_S$. ◁

We next show that M_S obeys properties (1) and (2) given in the lemma statement.

(1) Since every edge in M_S that is not incident to any ℓ-vertex is of the type either (u_+, v_-) or (u_0, v_0), M_S avoids all edges between a subscript 0 vertex and a subscript $+$ vertex.

(2) For any vertex u unmatched in T_S, we have (by complementary slackness) $\alpha_u = \mathsf{wt}_{T_S}(u, u) = 0$, i.e., $u \in S$. Thus for every $u \in V \setminus S$, we have $(u, v) \in T_S$ for some $v \in \mathsf{Nbr}(u)$; if $\alpha_u = -1$ then $(u_-, v_+) \in M_S$ else $(u_-, \ell(u)) \in M_S$. Thus all vertices in $\{u_- : u \in V \setminus S\}$ are matched in M_S.

In order to show M_S is a desired stable matching in G_S, we need to show this claim.

▷ Claim 17. M_S is a stable matching in G_S.

Proof. By the definition of M_S, the vertices $\ell(u)$ for all $u \in V \setminus S$ are matched in M_S. Thus for any $u \in V \setminus S$, all of $u_+, u_-, \ell(u)$ are matched in M_S, so neither $(u_+, \ell(u))$ nor $(u_-, \ell(u))$ blocks M_S. Other than edges incident to dummy vertices, the graph G_S consists of edges of the type $(u_+, v_-), (u_0, v_0), (u_+, v_0)$, i.e., $\{\alpha_u, \alpha_v\}$ is one of $\{1, -1\}, \{0, 0\}, \{1, 0\}$.

So for every $(u,v) \in E$ such that (u_{s_u}, v_{s_v}) is in G_S, we have $\alpha_u + \alpha_v \leq 1$, i.e., $\mathsf{wt}_{T_S}(u,v) \leq 1$ which means $\mathsf{wt}_{T_S}(u,v) \leq 0$. The constraint $\mathsf{wt}_{T_S}(u,v) \leq 0$ implies one of the 3 possibilities: (i) $(u,v) \in T_S$, (ii) u prefers $T_S(u)$ to v, (iii) v prefers $T_S(v)$ to u. In case (i), we have $(u_{s_u}, v_{s_v}) \in M_S$ and in cases (ii) and (iii), one of u_{s_u}, v_{s_v} is matched in M_S to a more preferred neighbor in G_S. Thus M_S is a stable matching in G_S. ◁

Conversely, suppose G_S admits such a stable matching M_S. We will show a truly popular matching T_S in G with a desired witness $\vec{\alpha}$. The matching T_S is easy to define:

$$T_S = \{(u,v) : (u_0, v_0) \in M_S \text{ or } (u_+, v_-) \in M_S\}.$$

We now need to show that T_S is a truly popular matching in G. For this, we will show a witness $\vec{\alpha} \in \{0, \pm 1\}^n$. Define $\alpha_u = 0$ for all $u \in S$. We will now define α_u for each $u \in V \setminus S$.

For each $u \in V$, note that $\ell(u)$ is top choice for u_-: hence $\ell(u)$ always has to be matched in any stable matching in G_S. For each $u \in V \setminus S$:

$$\text{let } \alpha_u = \begin{cases} -1 & \text{if } (u_+, \ell(u)) \in M_S \\ 1 & \text{if } (u_-, \ell(u)) \in M_S. \end{cases}$$

Observe that all edges in M_S not involving any ℓ-vertex are of the form either (u_+, v_-) or (u_0, v_0). This is because M_S avoids all edges of the type (u_+, v_0) by property (1) of a desired stable matching. Thus $\alpha_u + \alpha_v = 0$ for all $(u,v) \in T_S$.

▷ Claim 18. For any vertex u left unmatched in T_S, we have $u \in S$, i.e., $\alpha_u = 0$.

Proof. Every vertex of the form u_+ (being the top choice vertex of $\ell(u)$) has to be matched in every stable matching in G_S; also, all vertices in $\{u_- : u \in V \setminus S\}$ are matched in M_S by property (2). Hence M_S matches u_+, u_- for all $u \in V \setminus S$; thus one of u_+, u_- has to be matched to a non-dummy neighbor, i.e., a vertex other than $\ell(u)$. Hence for any vertex u left unmatched in T_S, we have $u \in S$. ◁

We have $\sum_{u \in V} \alpha_u = \sum_{(u,v) \in T_S} (\alpha_u + \alpha_v)$ from Claim 18 and by definition, $\alpha_u + \alpha_v = 0$ for each $(u,v) \in T_S$. Hence $\sum_{u \in V} \alpha_u = 0$. Every vertex in $V \setminus S$ is matched in T_S (by Claim 18) and so we have $\alpha_u \geq -1 = \mathsf{wt}_{T_S}(u,u)$ for $u \in V \setminus S$. For any vertex $u \in S$, we have $\alpha_u = 0 \geq \mathsf{wt}_{T_S}(u,u)$. Thus $\alpha_u \geq \mathsf{wt}_{T_S}(u,u)$ for every vertex u.

It can also be shown that $\alpha_u + \alpha_v \geq \mathsf{wt}_{T_S}(u,v)$ for every edge (u,v) in G. Thus T_S is a truly popular matching in G and the theorem follows. ◀

All stable matchings in a roommates instance match the same subset of vertices [19]. Call these vertices *stable*. Our algorithm for deciding if G admits a truly popular matching (and returning one, if so) is as follows:

1. For each set $S \subseteq V$ do:
   - Build the graph G_S and check if (i) all subscript $-$ vertices are stable in G_S and (ii) G_S admits a stable matching M_S that satisfies property 1 given in Theorem 15; if so, then return the corresponding matching T_S in G.
2. Return "no".

If our algorithm returns a matching T_S in Step 1, then T_S is truly popular (by Theorem 15). Suppose the algorithm reaches Step 2: so there is no $S \subseteq V$ such that G_S admits a stable matching that satisfies property 1. Then G has no truly popular matching (by Theorem 15). Thus the correctness of our algorithm follows from Theorem 15.

Step 1, part (i) is implemented by running a stable matching algorithm (say, [24]) in G_S. Step 1, part (ii) is implemented by running the algorithm for finding a stable matching in a roommates instance with forbidden edges [14]. Since there are 2^n sets $S \subseteq V$, the running time of our algorithm is $O^*(2^n)$. Thus we have shown Theorem 3 stated in Section 1.

4.1 Proof of Lemma 14

We bound the number of special truly popular matchings in a graph H by bounding the number of stable matchings in some related graphs that we construct below. Let S_α be the set of special truly popular matchings in H with a specific witness $\vec{\alpha} \in \{\pm 1\}^t$, where t is the number of vertices in H. Define $\sigma \in \{\pm\}^t$ as follows: $\sigma_u = \mathsf{sign}(\alpha_u)$ for all vertices u in H where $\mathsf{sign}(\alpha_u) = +$ if $\alpha_u = 1$, else $\mathsf{sign}(\alpha_u) = -$.

Corresponding to $\sigma \in \{\pm\}^t$, we build the graph H_σ as follows. The vertex set of H_σ is $\{u_{\sigma_u} : u \text{ is a vertex in } H\}$. For each edge (u, v) in H where $\sigma_u = -$ and $\sigma_v = +$ do:

- if u prefers v to all its neighbors w in H with $\sigma_w = -$ then add the edge (u_-, v_+) to H_σ.

For any vertex u_{σ_u} in H_σ: u_{σ_u}'s preference order of neighbors in H_σ is as per u's preference order in H. Note that for any neighbor v_{σ_v} of u_{σ_u} in H_σ, we have $\sigma_v = +$ if $\sigma_u = -$ and vice-versa. This is because the edge set of H_σ consists only of edges of the type (a_-, b_+).

For each $M \in S_\alpha$, define $f_\alpha(M) = \{(u_{\sigma_u}, v_{\sigma_v}) : (u, v) \in M\}$. We show in Claim 19 below that for every $(u, v) \in M$, the edge $(u_{\sigma_u}, v_{\sigma_v})$ is in H_σ. Thus $f_\alpha(M)$ is a matching in H_σ. Moreover, $f_\alpha(M)$ is a stable matching in H_σ (see Claim 20). Note that f_α is one-to-one. Hence the total number of special truly popular matchings in H is at most the the maximum number of stable matchings in H_σ summed up over all $\sigma \in \{\pm\}^t$, or equivalently, over all $\vec{\alpha} \in \{\pm 1\}^t$. This sum is at most $c^t \cdot 2^t = (2c)^t$. ◀

▷ Claim 19. For every $(u, v) \in M$, the edge $(u_{\sigma_u}, v_{\sigma_v})$ is in H_σ.

Proof. We have $\alpha_u + \alpha_v = \mathsf{wt}_M(u, v) = 0$ (by complementary slackness) and so $\{\alpha_u, \alpha_v\} = \{-1, 1\}$. Assume without loss of generality that $\alpha_u = -1$ and $\alpha_v = 1$. So $\sigma_u = -$ and $\sigma_v = +$. For any neighbor w of u with $\sigma_w = -$, we have $\mathsf{wt}_M(u, w) \le \alpha_u + \alpha_w = -1 - 1 = -2$, i.e., both u and w prefer their partners in M to each other. Thus u prefers v to all its neighbors w in H with $\sigma_w = -$. Hence (u_-, v_+) is in H_σ. ◁

▷ Claim 20. $f_\alpha(M)$ is a stable matching in H_σ.

Proof. Every edge in H_σ is of the form (a_-, b_+) for some adjacent pair of vertices a, b in H and $\alpha_a = -1, \alpha_b = 1$. Since $\vec{\alpha}$ is a witness of M, we have $\mathsf{wt}_M(a, b) \le \alpha_a + \alpha_b = 0$. Thus either $(a_-, b_+) \in f_\alpha(M)$ or at least one of a, b is matched in M to a more preferred neighbor. So (a_-, b_+) does not block $f_\alpha(M)$. Thus $f_\alpha(M)$ has no blocking edge in H_σ. ◁

References

1 D. J. Abraham, R. W. Irving, T. Kavitha, and K. Mehlhorn. Popular Matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.

2 P. Biró, R. W. Irving, and D. F. Manlove. Popular matchings in the marriage and roommates problems. In *Proceedings of the 7th International Conference on Algorithms and Complexity (CIAC)*, pages 97–108, 2010.

3 F. Brandl and T. Kavitha. Two Problems in Max-Size Popular Matchings. *Algorithmica*, 81(7):2738–2764, 2019.

4 K.S. Chung. On the Existence of Stable Roommate Matchings. *Games and Economic Behavior*, 33(2):206–230, 2000.

5 M.-J.-A.-N. de C. (Marquis de) Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix.* L'Imprimerie Royale, 1785.

6 Condorcet method. `https://en.wikipedia.org/wiki/Condorcet_method`.

7 Á. Cseh, C.-C. Huang, and T. Kavitha. Popular matchings with two-sided preferences and one-sided ties. *SIAM Journal on Discrete Mathematics*, 31(4):2348–2377, 2017.

8 Á. Cseh and T. Kavitha. Popular edges and dominant matchings. *Mathematical Programming*, 172(1):209–229, 2018.

9 Á. Cseh and T. Kavitha. Popular Matchings in Complete Graphs. In *Proceedings of the 38th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 17:1–17:14, 2018.

10 W. H. Cunningham and A. B. Marsh. A primal algorithm for optimal matching. *Mathematical Programming*, 8:50–72, 1978.

11 B. C. Dean and S. Munshi. Faster algorithms for stable allocation problems. *Algorithmica*, 58(1):59–81, 2010.

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

13 Y. Faenza, T. Kavitha, V. Powers, and X. Zhang. Popular Matchings and Limits to Tractability. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2790–2809, 2019.

14 T. Fleiner, R. W. Irving, and D. F. Manlove. Efficient algorithms for generalised stable marriage and roommates problems. *Theoretical Computer Scienc*, 381:162–176, 2007.

15 F. V. Fomin and D. Kratsch. *Exact exponential algorithms.* Springer-Verlag New York, Inc., New York, 2010.

16 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

17 P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Science*, 20(3):166–173, 1975.

18 S. Gupta, P. Misra, S. Saurabh, and M. Zehavi. Popular matching in roommates setting is NP-hard. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2810–2822, 2019.

19 D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms.* MIT Press, Boston, MA, 1989.

20 M. Hirakawa, Y. Yamauchi, S. Kijima, and M. Yamashita. *On The Structure of Popular Matchings in The Stable Marriage Problem - Who Can Join a Popular Matching?* In the 3rd International Workshop on Matching Under Preferences (MATCH-UP), 2015.

21 C.-C. Huang and T. Kavitha. Near-popular matchings in the Roommates problem. *SIAM Journal on Discrete Mathematics*, 27(1):43–62, 2013.

22 C.-C. Huang and T. Kavitha. Popular matchings in the stable marriage problem. *Information and Computation*, 222:180–194, 2013.

23 C.-C. Huang and T. Kavitha. Popularity, mixed matchings, and self-duality. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2294–2310, 2017.

24 R.W. Irving. An efficient algorithm for the stable roommates problem. *Journal of Algorithms*, 6:577–595, 1985.

25 A. R. Karlin, S. Oveis Gharan, and R. Weber. A simply exponential upper bound on the maximum number of stable matchings. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 920–925, 2018.

26 T. Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43(1):52–71, 2014.

27 T. Kavitha. Popular half-integral matchings. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 22.1–22.13, 2016.

28 T. Kavitha, J. Mestre, and M. Nasre. Popular Mixed Matchings. *Theoretical Computer Science*, 412(24):2679–2690, 2011.

29 E. McDermid and R. W. Irving. Sex-equal stable matchings: Complexity and exact algorithms. *Algorithmica*, 68:545–570, 2014.

30 A. Subramanian. A New Approach to Stable Matching Problems. *SIAM Journal on Computing*, 23(4):671–700, 1994.

31 C.-P. Teo and J. Sethuraman. The geometry of fractional stable matchings and its applications. *Mathematics of Operations Research*, 23(4):874–891, 1998.

32 E. G. Thurber. Concerning the maximum number of stable matchings in the stable marriage problem. *Discrete Mathematics*, 248(1-3):195–219, 2002.

The Complexity of Finding S-Factors in Regular Graphs

Sanjana Kolisetty
Departments of Mathematics and EECS, CSE Division,
University of Michigan, Ann Arbor, MI, USA
sanjanak@umich.edu

Linh Le
Departments of Mathematics and EECS, CSE Division,
University of Michigan, Ann Arbor, MI, USA
likle@umich.edu

Ilya Volkovich
Department of EECS, CSE Division, University of Michigan, Ann Arbor, MI, USA
ilyavol@umich.edu

Mihalis Yannakakis
Department of Computer Science, Columbia University, New York, NY, USA
mihalis@cs.columbia.edu

Abstract

A graph G has an *S-factor* if there exists a spanning subgraph F of G such that for all $v \in V : \deg_F(v) \in S$. The simplest example of such factor is a 1-factor, which corresponds to a perfect matching in a graph. In this paper we study the computational complexity of finding S-factors in regular graphs. Our techniques combine some classical as well as recent tools from graph theory.

2012 ACM Subject Classification Mathematics of computing → Matchings and factors; Theory of computation → Problems, reductions and completeness

Keywords and phrases constraint satisfaction problem, Dichotomy theorem, Graph Factors, Regular Graphs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.21

Funding Research partially supported by the NSF-CCF AitF 1535912 Grant.
Sanjana Kolisetty: Research partially supported by the NSF-CCF AitF 1535912 Grant.
Linh Le: Research partially supported by the NSF-CCF AitF 1535912 Grant.

Acknowledgements The authors would like to thank the anonymous referees for their detailed comments and suggestions on the previous version of the paper.

1 Introduction

The *Constraint Satisfaction Problem* (CSP for short) has been a classical topic in computer science of both theoretical and practical importance. While CSPs can be quite general, in this paper we focus on the "fixed-template" Boolean CSPs. That is, CSPs over the Boolean domain where the constraints come from a fixed set of Boolean relations Γ. Formally, given a fixed set of Boolean relations $\Gamma = \{R_1, R_2, \ldots, R_m\}$, a Γ-*formula* is a conjunction of constraints of the form $R_j(x_{i_1}, \ldots, x_{i_n})$ where $R_j \in \Gamma$ and the x_{i_j}-s are propositional variables; CSP(Γ) forms a decision problem where one needs to determine if a given Γ-formula is satisfiable. In other words, one needs to determine whether it is possible to satisfy all the constraints as given by the relations from Γ simultaneously.

The object of study is the computational complexity of CSP(Γ) as per the choice of Γ. In a seminal work of [22], Schaefer identified six classes of sets of Boolean relations for which CSP(Γ) $\in$ P and proved that all other sets of relations generate an NP-complete problem. This result is what is known as **Schaefer's Dichotomy Theorem** which provides a complete classification of the computational complexity of CSP(Γ). The two most popular examples of applications of this theorem are the NP-completeness of the 1-*in*-3SAT and *not-all-equal* 3SAT (NAE-3SAT) problems. Subsequently, in [3], a more refined classification was presented.

While a more general Dichotomy Theorem was recently proved for non-Boolean CSPs [6, 24][1], there has been a large body of work dedicated to the study of the computational complexity of a restricted version of CSP(Γ), denoted as $\mathrm{CSP}_2(\Gamma)$ or $\mathrm{CSP}_{\mathrm{Edge}}(\Gamma)$ [15, 10, 7, 14, 11, 8, 17]. Formally introduced by Feder in [10], $\mathrm{CSP}_2(\Gamma)$ corresponds to a specialization of CSP(Γ) to the instances where each variable appears at most twice. Alternatively, one can think about embedding the input Γ-formula into a graph, such that edges correspond to variables and nodes to constraints, and the constraint satisfaction problem asks for a spanning subgraph such that the set of its edges at each node satisfies the constraint at the node.

The main subtlety is that if CSP(Γ) $\in$ P then, clearly, $\mathrm{CSP}_2(\Gamma) \in$ P. However, in the general NP-hard instances there is usually no restriction of the number of appearances of a variable. Therefore, a proof that CSP(Γ) is NP-hard may not carry over to $\mathrm{CSP}_2(\Gamma)$. In particular, if we consider the aforementioned examples, CSP_2(1-in-3SAT) corresponds to determining existence of a perfect matching in a 3-regular graph, which is decidable in polynomial time. In addition, CSP_2(NAE-3SAT) is "trivial" since every read-twice[2] NAE-3SAT-formula is always satisfiable![3]

Despite all the invested effort, we are still far away from the ultimate goal. Indeed, the known results do not even provide a complete classification for the cases when Γ consists of just a single relation! A natural focus taken in [15], was to consider the sets Γ that consist of symmetric relations as these instances often arise more naturally in the graph context, because incident edges to a node are typically treated symmetrically in graph theory. This class of problems can be regarded as generalized matchings. In this paper we give a complete classification of the computational complexity of $\mathrm{CSP}_2(\Gamma)$, where Γ consists of a single symmetric relation.

In turns out that the problem has a very natural interpretation in terms of finding "S-factors" of regular graphs. We say that a graph G has an *S-factor*, if there exists a spanning subgraph F of G such that for all $v \in V : \deg_F(v) \in S$. The simplest example of such factor is a 1-factor, which corresponds to a perfect matching in a graph.

1.1 Results

In light of the natural interpretation of the problem in terms of finding S-factors of graphs, we present our main results using that terminology. The CSP versions of the main results (and their proofs) can be found in Section 4. Our first result is a Dichotomy Theorem for regular graphs of even degree. The Dichotomy is obtained by classifying all the tractable cases.

[1] Affirming what was known as the CSP Dichotomy Conjecture formulated in [12].

[2] A formula in which every variable appears at most twice.

[3] This follows immediately from Tutte's Theorem (Lemma 8), however there is a more direct way to see that.

▶ **Theorem 1.** *Let $\ell \in \mathbb{N}$. There is a polynomial-time algorithm that given a 2ℓ-regular graph G as an input, finds an S-factor in G, if there is one, in the following four cases:*

1. *S contains an even number.*
2. *$\ell \in S$.*
3. *$\{\ell - 1, \ell + 1\} \subseteq S$.*
4. *$S = \{p, p+2, \cdots, p+2r\}$ for some $p, r \geq 0$.*

Otherwise, finding an S-factor is NP*-Hard.*

As could be observed, all the tractable cases reduce to the case of finding a perfect matching in a graph (see Section 2.1 for details). Consequently, an algorithm for perfect matching in graphs (e.g. [9]) could be used to find these S-factors, for the "yes"-instances of the problem. For regular graph of odd degree, we obtain a somewhat weaker result: we show that for each set S, the decision problem is either polynomial-time solvable or NP-hard, yet we are unable to classify explicitly all the tractable cases. Closing this gap will require resolving several conjectures in graph theory (see [2, 19, 1, 5] for more details).

▶ **Theorem 2.** *Let $\ell \in \mathbb{N}$. There is a polynomial-time algorithm that given a $(2\ell + 1)$-regular graph G as an input, decides if G has an S-factor, in the following two cases:*

1. *Every $(2\ell + 1)$-regular graph has an S factor.*
2. *$S = \{p, p+2, \cdots, p+2r\}$ for some $p, r \geq 0$.*

Otherwise, deciding if G has an S-factor is NP*-Hard.*

There are specific sets S, for which it is an open problem in graph theory whether every $(2\ell + 1)$-regular graph has an S-factor. A simple concrete example is the case of $S = \{1, 4\}$ for degree-5 graphs (the conjecture in this case is that there is always an S-factor). The theorem tells us that, even though we may not know the answer to the open problem for a particular S, if it does not hold trivially for all graphs and there is a counterexample, then the corresponding S-factor problem is NP-hard; that is, there is a way to use any counterexample (as a black box) to generate an NP-hardness reduction.

1.2 Comparison to Previous Results

In [15], Istrate studied the special case when Γ consists of symmetric relations. In that work, several "patterns" for which $\mathrm{CSP}_2(\Gamma) \in \mathsf{P}$ were identified. In particular, one such pattern corresponds to Case 4 of Theorem 1. This result was obtained via connections to covering problems. In addition, Istrate formulated a sufficient condition under which the computational complexity of $\mathrm{CSP}_2(\Gamma)$ and $\mathrm{CSP}(\Gamma)$ is the same, with the additional "constants for free" assumption. That is, one can fix some variables to either 0 and 1 (for more details, see Lemma 26 and the preceding discussion). Later on, Feder [10], extended the condition to non-symmetric relations, introducing *Delta Matroids*, and showed that if Γ contains some relation that is not a Delta matroid then $\mathrm{CSP}_2(\Gamma)$ and $\mathrm{CSP}(\Gamma)$ have the same complexity (in the presence of constants). Several subsequent works [7, 8, 17] introduced further refinements to Delta Matroids. Yet, "constants for free" remained a prevalent assumption in these and other CSP-related works. Nonetheless, even with the assumption, no classification for the mere case of a single symmetric relation was known prior to our work.

We also would like to point out that the "constants for free" assumption is implicitly equivalent to adding two more relations $P(x) = x$ and $Q(x) = \neg x$ to Γ. It is important to stress that adding these relations can completely tilt the scale. For example, consider a single 8-ary symmetric relation "two or six out of eight". Formally, $R(\bar{x}) = 1$ iff $w_H(x) = 2$ or 6. In the graphical perspective, this corresponds to the problem of finding a $\{2, 6\}$-factor of an 8-regular graph. Now by TutteŠs Theorem (Lemma 8), every 8-regular graph has a 2-factor. Hence $\mathrm{CSP}_2(R) \in \mathsf{P}$ in a "trivial" way. On the other hand, $\mathrm{CSP}_2(\{R, x, \neg x\})$ is NP-hard (follows e.g. from [15]). Our results do not rely on the “constants for free” assumption. In fact, they complement it: roughly speaking, we show that either $\mathrm{CSP}_2(\Gamma) \in \mathsf{P}$ or there exist Γ-formulas that “implement” the relations x and $\neg x$. See Lemmas 29 and 30 for more details.

There is, of course, extensive work in graph theory on factors in graphs, (see e.g. the surveys [2, 21] and references therein), with the development of a rich theory of matchings, as well as more general factors. This includes structural results on the existence of factors, starting from Petersen’s theorem from 1891 [20]; algorithmic results, including e.g. Edmonds’ matching algorithm [9] and its extensions and refinements; and hardness results, starting e.g. with Lovász’s theorem [18] that for any $a, b \in \mathbb{N}$ such that $1 \le a \le b - 3$, the problem of deciding whether a graph has an $\{a, b\}$-factor is NP-hard even for simple graphs (not necessarily of a given, regular degree). We will leverage several of these graph theoretic results on (generalized) matchings and the existence of suitable factors in graphs. We review some of these theorems that we use in the next section.

2 Preliminaries

▶ **Definition 1** (Zebras and Holes). *Let $S \subseteq \mathbb{N}$ be a subset of $\mathbb{N}$. Following [15], we say that S contains a* hole of size t *if there exist i such that: $i, i+t+1 \in S$ and $[i+1, i+t] \cap S = \emptyset$. Let $a \le b \in \mathbb{N}$ such that $a \equiv b (\bmod 2)$. We say that S is an (a, b)-*zebra *if $S = \{a, a+2, a+4, \dots, b\}$. We call a set S a zebra, if it is an (a, b)-zebra for some $a, b \in \mathbb{N}$.*

▶ Remark. A set $S = \{a\}$ also constitutes a zebra since it is an (a, a)-zebra. The following is a simple observation about the structure of finite subsets of $\mathbb{N}$, that will be useful for us later.

▶ **Observation 2.** *Let $S \subseteq \mathbb{N}$ be a finite, non-empty subset of $\mathbb{N}$ then (at least) one of the following holds:*

- *S contains two consecutive numbers.*
- *S is zebra.*
- *S contains a hole of size at least 2.*

2.1 Graphs

In this paper we consider graphs $G = (V, E)$. Unless specified otherwise, all the graphs considered in the paper are general graphs (i.e. with self-loops and parallel edges). The focus of this paper is the complexity of finding a particular kinds of subgraphs in graphs, known as *factors*. We define this formally now.

▶ **Definition 3** (Factors). *Let $G = (V, E)$ be a graph with V vertices and E edges. [2]*

1. *H-factor: Let H be a set function associated with G that maps $V \to 2^{\mathbb{N}}$. We say that G has an H-*factor *if there exists a spanning subgraph F of G such that for all $v \in V : \deg_F(v) \in H(v)$.*
2. *f-factor is a specialization to the case when $\forall v \in V : H(v) = \{f(v)\}$, for some function $f : V(G) \to \mathbb{Z}_+$.*
3. *S-factor is a specialization to the case when $H(v) = S$ for all $v \in V$, for some* fixed *set $S \subseteq \mathbb{N}$.*
4. *$[a, b]$-factor is a further specialization to the case when S is the interval $[a, b]$.*
5. *k-factor is a further specialization to the case when $S = \{k\}$.*

The simplest case of a graph factor is a 1-factor which corresponds to a perfect matching of a graph. This problem has a well-known efficient algorithm known as “Blossom Algorithm”.

▶ **Lemma 4** ([9]). *There exists a polynomial-time algorithm that given a graph $G = (V, E)$ as an input, outputs a 1-factor F of G, if one exists.*

The algorithm can be easily extended to handle f-factors due to the following observation:

▶ **Lemma 5** ([4]). *There exists a polynomial-time algorithm that given a graph $G = (V, E)$ and a function $f : V \to \mathbb{Z}$ (as a vector) as an input, outputs a graph G' such that G' has a 1-factor F' iff G has an f-factor F. Moreover, F can be computed in polynomial time given F'.*

In addition, using the simple idea of [16] of introducing self-loops, the algorithm can be further extended to H-factors, where each $H(v)$ is a zebra (See Definition 1).

▶ **Lemma 6** ([16]). *There exists a polynomial-time algorithm that given a graph $G = (V, E)$ and a function $H : V \to 2^{\mathbb{N}}$, where each $H(v)$ is zebra, as an input, outputs a graph G' and a function $f : V \to \mathbb{Z}$ such that G' has a f-factor F' iff G has an H-factor F. Moreover, F can be computed in polynomial time given F'.*

The following is immediate given the above reductions to the perfect matching case.

▶ **Corollary 7.** *There exists a polynomial-time algorithm that given a graph $G = (V, E)$ and a function $H : V \to 2^{\mathbb{N}}$, where each $H(v)$ is zebra, as an input, outputs an H-factor F of G, if one exists.*

We note that an efficient algorithm for this kind of H-factors has been obtained in [15] using a different argument. Recently in [17], the algorithm was extended to also handle the "asymmetric" version. Next, we require the following results regarding the existence of regular factors in regular graphs.

▶ **Lemma 8** (Regular Factors of Regular Graphs).

1. [23] *Let $r, k \in \mathbb{N}$ such that $1 \leq k \leq r-1$. Then any r-regular graph has a $\{k, k+1\}$-factor.*
2. [20] *Let r and k be even integers such that $2 \leq k \leq r$. Then any r-regular graph has a k-factor.*
3. [13] *Suppose r is even and $\frac{r}{2}$ is odd. Then any connected r-regular graph of even order has a $\frac{r}{2}$-factor.*

As a corollary we obtain the following, which was observed for simple graphs in [1]. We also note that the proof of [1] is merely existential whereas our proof is algorithmic.

▶ **Lemma 9.** *Let $r \in \mathbb{N}$ such that both r and $\frac{r}{2}$ are even. Then any r-regular graph of even order has a $\left\{\frac{r}{2} - 1, \frac{r}{2} + 1\right\}$-factor.*

Proof. Let $G = (V, E)$ be a graph satisfying the preconditions. For every $v \in V$, we add a self-loop. Call this new resulting graph $G' = (V, E')$. Observe that G' is a $(r + 2)$-regular graph of even order and $\frac{r+2}{2} = \frac{r}{2} + 1$ is odd. Therefore, by Lemma 8, G' has a $(\frac{r}{2} + 1)$-factor. Now, consider two cases: if $v \in V$ uses the self-loop to fulfill its factor, then the induced degree of v in G is $\frac{r}{2} - 1$. Otherwise, the induced degree of v in G is $\frac{r}{2} + 1$. ◀

2.2 Boolean Relations

▶ **Definition 10.** *The* Hamming Weight *of a vector $\bar{v} \in \{0, 1\}^n$ is defined as:* $\mathrm{w_H}(\bar{v}) \triangleq |\{i \mid v_i \neq 0\}|$. *That is, the number of its non-zero coordinates.*

▶ **Definition 11** (Symmetric Relation). *We say that an m-ary relation $R(x_1, x_2, \ldots, x_m)$ is* symmetric *if there exists a set* $\mathsf{Spec}(R) \subseteq \{0 \ldots m\}$ *such that* $R(\bar{x}) = 1$ *if and only if* $\mathrm{w_H}(\bar{x}) \in \mathsf{Spec}(R)$*. The set* $\mathsf{Spec}(R)$ *is called the* spectrum *of R.*

The following are examples of particular symmetric relations we will be utilizing.

▶ **Example 12.**

- $\mathrm{NE}(x_1, x_2)$ is a binary relation with $\mathsf{Spec}(\mathrm{NE}) = \{1\}$.
- Let $k \in \mathbb{N}$. EQ_k is a k-ary relation with $\mathsf{Spec}(\mathrm{EQ}_k) = \{0, k\}$.

▶ **Definition 13** (Dual Relation). *Let $R(x_1, \ldots, x_m)$ be a relation. We define the* dual relation *of R as:*

$$R^*(x_1, \ldots, x_m) \triangleq R(\neg x_1, \neg x_2, \ldots, \neg x_m).$$

The following observation is immediate with respect to symmetric relations.

▶ **Observation 14.** *For a symmetric m-ary relation R we have:* $\mathsf{Spec}(R^*) = \{m - i \mid i \in \mathsf{Spec}(R)\}$.

2.2.1 Γ-Instances, CSP(Γ), Triviality

In what follows, let $\Gamma = \{R_1, R_2, \ldots, R_\ell\}$ be a *fixed* set of Boolean relations. We will use Γ^* to denote the set of dual relations. Formally, $\Gamma^* \triangleq \{R_1^*, R_2^*, \ldots, R_\ell^*\}$, where R_i^* is the dual relation of R_i.

▶ **Definition 15.** *A* Γ-instance *or* Γ-formula Φ *is a conjunction of constraints of the form $R_j(x_{i_1}, \ldots, x_{i_n})$ where $R_j \in \Gamma$ and the x_{i_j}-s are propositional variables. The* read *of a variable x_i in Φ is the number of occurrences of x_i in Φ. The* read *of a formula Φ is the maximal read of a variable in it.*

In this paper we will focus on read-twice formulas, that is formulas in which all the variables appear at most two times. We now formally introduce the main problem we will study.

▶ **Problem 16.** $\mathrm{CSP}(\Gamma)$ *forms a decision problem where one needs to determine if a given Γ-formula is satisfiable. In other words, one needs to determine whether it is possible to satisfy **all** the constraints as given by the relations from Γ, simultaneously. For $k \geq 1$,* $\mathrm{CSP}_k(\Gamma)$ *is a specialization of* $\mathrm{CSP}(\Gamma)$ *to read-k instances. If $\Gamma = \{R\}$ has a single relation R, we will write* $\mathrm{CSP}(R)$ *and* $\mathrm{CSP}_k(R)$.

As was pointed out in the introduction, in this paper we are interested in the computational complexity of $\mathrm{CSP}_2(\Gamma)$, as per the choice of Γ. We now recall Schaefer's Dichotomy Theorem [22].

▶ **Lemma 17** ([22]). $\mathrm{CSP}(\Gamma) \in \mathsf{P}$ *in the following six cases:*

1. $\forall R_j \in \Gamma : R_j(\bar{0}) = 1$
2. $\forall R_j \in \Gamma : R_j(\bar{1}) = 1$
3. $\forall R_j \in \Gamma : R_j$ *is equivalent to a conjunction of binary relations*
4. $\forall R_j \in \Gamma : R_j$ *is equivalent to a conjunction of Horn clauses*
5. $\forall R_j \in \Gamma : R_j$ *is equivalent to a conjunction of dual-Horn clauses*
6. $\forall R_j \in \Gamma : R_j$ *is equivalent to a conjunction of affine forms*

Otherwise, $\mathrm{CSP}(\Gamma)$ *is* NP*-Hard.*

The following is an instantiation of the Theorem to the case of a single symmetric relation.

▶ **Corollary 18.** *Let $R(x_1, \ldots, x_m)$ be a symmetric relation. Then* $\mathrm{CSP}(R) \in \mathsf{P}$ *in the following cases:*
1. $R(\bar{0}) = 1$
2. $R(\bar{1}) = 1$
3. $m \leq 2$
4. $\mathsf{Spec}(R)$ *contains all odd numbers in* $\{1, \ldots, m\}$

▶ **Definition 19** (Triviality). *We say that* $\mathrm{CSP}_k(\Gamma)$ *is* trivial *if every instance of* $\mathrm{CSP}_k(\Gamma)$ *(i.e. every read-k Γ-instance) is satisfiable.*

To put the above definitions into a context, observe the first two of the six tractable classes correspond to cases when $\mathrm{CSP}(\Gamma)$ and $\mathrm{CSP}(\Gamma^*)$ are trivial. Similarly, observe that Cases 4 and 5 correspond the same conditions applied to both $\mathrm{CSP}(\Gamma)$ and $\mathrm{CSP}(\Gamma^*)$. With some extra work, you can see that the same holds true for Cases 3 and 6. In the same vein, the following lemma is immediate from the definition.

▶ **Lemma 20.**
1. $\mathrm{CSP}_1(\Gamma)$ *is trivial, as long as* Γ *does not contain a contradiction.*
2. *For any $k \in \mathbb{N}$ and any Γ:* $\mathrm{CSP}_k(\Gamma)$ *is trivial iff* $\mathrm{CSP}_k(\Gamma^*)$ *is trivial.*

2.2.2 Induced Relations and Implementation

▶ **Definition 21** (Induced relation). *For a relation $R(\bar{x}, \bar{y})$ we define the* induced relation *$\exists \bar{y} R(\bar{x}, \bar{y})$ on $\bar{x}$ as*

$$\exists \bar{y} R(\bar{x}, \bar{y}) = 1 \iff \exists \bar{y} \text{ such that } R(\bar{x}, \bar{y}) = 1.$$

▶ **Definition 22** (Implementation). *Let $R(\bar{x})$ be an arbitrary relation. We say that Γ* implements *R, denoted by Γ imp R, if there exists a Γ-instance $\Phi(\bar{x}, \bar{y})$ such that $R(\bar{x}) = \exists \bar{y} \Phi(\bar{x}, \bar{y})$. Furthermore, we say that Γ* read-twice-implements *R, denoted by Γ imp_2 R, if in addition:*
1. *Each x_i is read-once in Φ.*
2. *Each y_j is (at most) read-twice in Φ.*

The intuition behind the definition is that if Γ read-twice-implements R then we can, effectively, consider the set $\Gamma \cup \{R\}$ instead of Γ. The following lemma summarizes this intuition and will be used implicitly in our proofs.

▶ **Lemma 23.** *Let R be a relation such that Γ read-twice-implements R and let $\Gamma' \triangleq \Gamma \cup \{R\}$. Then for every read-twice Γ'-instance $\Phi'(\bar{x})$ there exists a read-twice Γ-instance $\Phi(\bar{x}, \bar{y})$ such that $\Phi'(\bar{x}) = \exists \bar{y} \Phi(\bar{x}, \bar{y})$.*

The following lemma showcases this intuition further, by showing that a three-way Equality EQ_3 can be used to implement k-way equality EQ_k for any $k \geq 3$. Conversely, one can use k-way equality EQ_k to implement k'-way equality $\mathrm{EQ}_{k'}$ for any $k' \leq k$.

▶ **Lemma 24.** *If Γ read-twice-implements EQ_3 then Γ read-twice-implements EQ_k, for any $k \geq 3$.*

Proof. By induction on k. The base case $k = 3$ is trivial. Let Φ_k denote a Γ-instance that read-twice implements $\mathrm{EQ}_k(x_1, \ldots, x_k)$. Given Φ_k, we can read-twice implement $\mathrm{EQ}_{k+1}(x_1, \ldots, x_{k+1})$ in the following way:

$$\Phi_k(x_1, \ldots, x_{k-1}, y) \wedge \Phi_3(y, x_k, x_{k+1}).$$

Given our inductive hypothesis and the fact that we can read-twice implement EQ_3, we can conclude that Γ read-twice-implements EQ_{k+1}. ◀

We note this was already observed in [15, 10]. Similar ideas can be used to show that if Γ read-twice-implements particular relations, then read-twice Γ-formulas exhibit some interesting closure properties.

▶ **Lemma 25** (Read-Twice Implementing Particular Relations).

1. ***Closure Under Variable Negation:*** *Suppose* Γ *read-twice-implements* NE. *Then read-twice* Γ*-formulas are closed under variable negation. Formally, if* $\Gamma \text{ imp}_2 R(x, \bar{y})$ *then* $\Gamma \text{ imp}_2 R(\neg x, \bar{y})$*.*
2. ***Closure Under Setting Variables to Constants:*** *Suppose* Γ *read-twice-implements* x *or* $\neg x$*. Then read-twice* Γ*-formulas are closed under setting variables to either* 1 *or* 0*, respectively. Formally, if* $\Gamma \text{ imp}_2 R(x, \bar{y})$ *then* $\Gamma \text{ imp}_2 R(1, \bar{y})$ *or* $R(0, \bar{y})$*, respectively.*
3. ***Closure Under Variable Repetition:*** *Suppose* Γ *read-twice-implements* EQ_k*. Then read-twice* Γ*-formulas are closed under repetition of any variable arbitrary number of times.*

We will use the above implicitly. We finish this section with the following simple observation from [15].

▶ **Lemma 26** ([15]). *Let* R *be a symmetric relation such that* Spec(R) *contains a hole of size at least* 2*. Then* $\{R, x, \neg x\}$ *read-twice-implements* EQ_3*.*

We note that Feder [10] extended this claim to a non-symmetric case defining *Delta Matroids*. In the same paper it was observed that WLOG every variable in a read-twice formula occurs *exactly* twice. Furthermore, such formulas have very natural interpretation as graphs where edges play the role of variables and nodes the role of constraints.

▶ **Lemma 27** (Graphical Perspective of CSP_2 [10]). *Every read-twice formula can be efficiently transformed into an exact read-twice formula, and furthermore viewed as a graph.*

3 Main Technical Tools

In this section we present our main technical tools, which we will use to prove Theorems 1 and 2. We begin by showing that the sets Γ for which $\text{CSP}_2(\Gamma)$ is non-trivial (in the sense of Definition 19), read-twice implement $\text{NE}(x, y)$ or $\{x, \neg x\}$. Consequently, by Lemma 25, such read-twice Γ-formula are closed under variable negation or setting variables to constants $\{0, 1\}$. Note that the result holds for general relations (not necessarily symmetric).

▶ **Lemma 28.** *Suppose that* $\text{CSP}_2(\Gamma)$ *is non-trivial. Then* Γ *read-twice-implements* $\text{NE}(x, y)$ *or* $\{x, \neg x\}$*.*

Proof. Since $\text{CSP}_2(\Gamma)$ is non-trivial, there exists an unsatisfiable read-twice Γ-instance Φ. On the other hand, recall (e.g. Lemma 20) that any read-once Γ-instance is satisfiable. Consider the "unpaired" version Φ' of Φ. Formally, for each variable x_i we replace one of the occurrences with a fresh new variable y_i. Observe that the resulting Φ' is read-once and hence satisfiable. Now, consider the process of gradually pairing the variables of Φ', that will eventually recover Φ. Formally, $\Phi'_0 \triangleq \Phi'$. Φ'_1 results from Φ'_0 by setting $x_1 = y_1$. More generally, Φ'_i results from Φ'_{i-1} by setting $x_i = y_i$. As $\Phi'_0 = \Phi'$ is satisfiable and $\Phi'_n = \Phi$ is not, by a hybrid argument, there exist i such that Φ'_{i-1} is satisfiable and Φ'_i is not. Let $\varphi(x_i, y_i)$ be the relation given by Φ'_{i-1} induced to the variables x_i and y_i (Recall Definition 21). By the above, $\varphi(0, 0) = \varphi(1, 1) = 0$ and either $\varphi(0, 1) = 1$ or $\varphi(1, 0) = 1$ (or both). Consider three cases:

- $\varphi(0,1) = \varphi(1,0) = 1$. In this case: $\varphi(x_i, y_i) = \mathrm{NE}(x_i, y_i)$.
- $\varphi(0,1) = 0, \varphi(1,0) = 1$. In this case: $\exists y_i \varphi(x_i, y_i) = x_i$ and $\exists x_i \varphi(x_i, y_i) = \neg y_i$.
- $\varphi(0,1) = 1, \varphi(1,0) = 0$. In this case: $\exists y_i \varphi(x_i, y_i) = \neg x_i$ and $\exists x_i \varphi(x_i, y_i) = y_i$. ◀

Next, we show that for symmetric relations we can derive further closure properties under some technical conditions.

▶ **Lemma 29.** *Let R be a symmetric 2ℓ-ary relation such that: $\ell \notin \mathsf{Spec}(R)$ and $\{\ell - 1, \ell + 1\} \not\subseteq \mathsf{Spec}(R)$. Then $\{R, \mathrm{NE}\}$ read-twice-implements EQ_3 or $\{x, \neg x\}$.*

Proof. We define the following two sets: $S_- = \{a \mid R(\ell - a) = 1\}$ and $S_+ = \{a \mid R(\ell + a) = 1\}$. Furthermore, let $a_- = \min S_-$ and $a_+ = \min S_+$. We define a_- or a_+ to be infinity if S_- or S_+ is empty, respectively. We consider three cases:

- **Case 1:** $a_+ = a_-$. Observe that $a_- \geq 2$. Using NE and Lemma 25, we plug $\ell - a_-$ pairs $z_i, \neg z_i$ into the relation R. Formally, consider,

  $$R(\bar{z}, \bar{y}) \triangleq R(z_1, \neg z_1, \ldots, z_{\ell - a_-}, \neg z_{\ell - a_-}, y_1, \ldots, y_{2a_-}).$$

  By definition, $\mathrm{w_H}(\bar{z}) = \ell - a_-$ and $0 \leq \mathrm{w_H}(\bar{y}) \leq 2a_-$. Now, since $a_+ = a_-$:

  $$R(\bar{z}, \bar{y}) = 1 \iff \mathrm{w_H}(\bar{y}) \in \{0, 2a_-\}.$$

  Consequently, $\exists \bar{z} R(\bar{z}, \bar{y}) = \mathrm{EQ}_k(\bar{y})$, where $k = 2a_- \geq 4$.
- **Case 2:** $a_+ > a_-$. Observe that $a_- \geq 1$ and consider $R(\bar{z}, \bar{y})$ as above. Now, however, since $a_+ > a_-$:

  $$R(\bar{z}, \bar{y}) = 1 \iff \mathrm{w_H}(\bar{y}) = 0.$$

  Hence, we obtain $\neg y_i$. Using NE, we can obtain y_i.
- **Case 3:** $a_- > a_+$. Observe that $a_+ \geq 1$. We repeat the argument of Case 2 for the dual relation R^* of R. As R^* read-twice-implements $\{x, \neg x\}$, so does R. ◀

We use a similar argument for relations of odd arity.

▶ **Lemma 30.** *Let R be a symmetric $2\ell + 1$-ary relation such that: $\{\ell, \ell + 1\} \not\subseteq \mathsf{Spec}(R)$. Then $\{R, \mathrm{NE}\}$ read-twice-implements EQ_3 or $\{x, \neg x\}$.*

Proof. We define the following two sets: $S_- = \{a \mid R(\ell - a) = 1\}$ and $S_+ = \{a \mid R(\ell + 1 + a) = 1\}$. Furthermore, let $a_- = \min S_-$ and $a_+ = \min S_+$. We define a_- or a_+ to be infinity if S_- or S_+ is empty, respectively. We consider three cases:

- **Case 1:** $a_+ = a_-$. Observe that $a_- \geq 1$. Consider,

  $$R(\bar{z}, \bar{y}) \triangleq R(z_1, \neg z_1, \ldots, z_{\ell - a_-}, \neg z_{\ell - a_-}, y_1, \ldots, y_{2a_- + 1}).$$

  By definition, $\mathrm{w_H}(\bar{z}) = \ell - a_-$ and $0 \leq \mathrm{w_H}(\bar{y}) \leq 2a_- + 1$. Now, since $a_+ = a_-$:

  $$R(\bar{z}, \bar{y}) = 1 \iff \mathrm{w_H}(\bar{y}) \in \{0, 2a_- + 1\}.$$

  Consequently, $\exists \bar{z} R(\bar{z}, \bar{y}) = \mathrm{EQ}_k(\bar{y})$, where $k = 2a_- + 1 \geq 3$.
- **Case 2:** $a_+ > a_-$. Observe that $a_- \geq 0$ and consider $R(\bar{z}, \bar{y})$ as above. Now, however, since $a_+ > a_-$:

  $$R(\bar{z}, \bar{y}) = 1 \iff \mathrm{w_H}(\bar{y}) = 0.$$

  Hence, we obtain $\neg y_i$. Using NE, we can obtain y_i.
- **Case 3:** $a_- > a_+$. Observe that $a_+ \geq 0$. We repeat the argument of Case 2 for the dual relation R^* of R. As R^* read-twice-implements $\{x, \neg x\}$, so does R. ◀

4 Characterization Proof

In this sections we give our main results, thus proving Theorems 1 and 2.

▶ **Theorem 31** (Characterization of Even-Arity Relations). *Let R be a symmetric 2ℓ-ary relation which is not constantly false. Then $\mathrm{CSP}_2(R) \in \mathsf{P}$ in the following four cases:*

1. *There is an even $k \in \mathsf{Spec}(R)$.*
2. *$\ell \in \mathsf{Spec}(R)$.*
3. *$\{\ell - 1, \ell + 1\} \subseteq \mathsf{Spec}(R)$.*
4. *$\mathsf{Spec}(R)$ is a zebra.*

Otherwise, $\mathrm{CSP}_2(R)$ is NP-Hard.

Proof. For Cases 1–4, we will take the graphical perspective (Lemma 27). Indeed, the problem corresponds to finding an S-factor of a given 2ℓ-regular graph, where $S = \mathsf{Spec}(R)$.

1. Follows from Item 2 of Lemma 8.
2. We can assume WLOG that S contains only odd numbers. In particular, ℓ is odd. Consider the following algorithm, given a graph G as an input.
   - Find all the connected components $C_1, C_2, \ldots, C_t$ of G.
   - If each C_i is of even order, return "true"; otherwise, return "false".

   Analysis: If each C_i is of even order, then by Lemma 8, each C_i has an ℓ-factor and so does G. Conversely, suppose some C_i is of odd order. Then by Handshaking Lemma, C_i cannot have an S-factor, as otherwise the overall sum of the degrees will be odd.
3. As before, we can assume WLOG that S contains only odd numbers. Hence, ℓ is even. Apply the procedure outlined in the proof of Lemma 9. This will reduce the problem to the previous case.
4. Apply Corollary 7 with $H(v) = \mathsf{Spec}(R)$ for every vertex v in the graph.

For the NP-Hardness proof, we take the CSP view of the problem. We show that if none of the Cases 1-4 hold, then $\mathrm{CSP}_2(R)$ is as hard as $\mathrm{CSP}(R)$. That is, we can lift the restriction on the read. The hardness then follows from Schaefer's Dichotomy Theorem instantiated to a single symmetric relation - Corollary 18.

▷ Claim 32. If $\mathsf{Spec}(R)$ does not fall into any of the four cases, then $\mathsf{Spec}(R)$ contains a hole of size at least 2 and $\{R\}$ read-twice-implements EQ_3.

Proof. First, observe that $\mathsf{Spec}(R)$ cannot have two consecutive numbers (as one of them will be even) and is not a zebra (Case 4). Therefore, by Observation 2, $\mathsf{Spec}(R)$ must contain a hole of size at least 2.
Next, consider the relation:

$$N(x, y) \triangleq \exists \bar{z} R(z_1, z_1, z_2, z_2, \ldots, z_{\ell-1}, z_{\ell-1}, x, y).$$

Since $\mathsf{Spec}(R)$ does not contain even numbers (Case 1), $N(x, y) = \mathrm{NE}(x, y)$. Thus, by Lemma 29 given Cases 2 and 3, $\{R\}$ read-twice-implements EQ_3 or $\{x, \neg x\}$. In the former case, the claim follows. In the latter case, Lemma 26 completes the proof of the claim. ◁

In conclusion, $\mathrm{CSP}_2(R)$ is as hard as $\mathrm{CSP}(R)$ and is thus NP-Hard by Corollary 18. ◀

For symmetric relations of odd arity, we obtain a somewhat weaker result.

▶ **Theorem 33** (Characterization of Odd-Arity Relations)**.** *Let R be a symmetric $(2\ell+1)$-ary relation which is not constantly false. Then* $\mathrm{CSP}_2(R) \in \mathsf{P}$ *in the following cases:*

1. $\mathrm{CSP}_2(R)$ *is trivial.*
2. $\mathsf{Spec}(R)$ *is a zebra.*

Otherwise, $\mathrm{CSP}_2(R)$ *is* NP*-Hard.*

Proof. Case 1 is trivial and Case 2 follows from Corollary 7.. For the NP-Hardness proof, we use a similar argument as in Theorem 31 to conclude that $\mathrm{CSP}_2(R)$ is as hard as $\mathrm{CSP}(R)$. Here is the high-level idea:

- $\{R\}$ read-twice-implements $\mathrm{NE}(x,y)$ or $\{x,\neg x\}$ - Lemma 28.
- $\mathsf{Spec}(R)$ cannot contain two consecutive numbers - Lemma 8.
- $\mathsf{Spec}(R)$ contains a hole of size at least 2 - Observation 2.
- $\{R\}$ read-twice-implements EQ_3 or $\{x,\neg x\}$ - Lemma 30.
- $\{R\}$ read-twice-implements EQ_3 - Lemma 26.

First observe that $\mathsf{Spec}(R)$ cannot contain two consecutive numbers since by Lemma 8, this case is trivial, in the graphical perspective. Consequently, by Observation 2, $\mathsf{Spec}(R)$ must contain a hole of size at least 2. In addition, by Lemma 30, $\{R\}$ read-twice-implements EQ_3 or $\{x,\neg x\}$. In the former case, we are done. In the latter case, Lemma 26 completes the proof. ◀

5 Discussion & Open Questions

In this paper we obtain the first classification of the computational complexity of $\mathrm{CSP}_2(R)$, where R is a single symmetric relation. Alternatively, we obtain a classification of the complexity of the S-factor problem for regular graphs. The characterization is explicit for even degree graphs (even arity), while for odd degrees it states that all nontrivial cases, except for zebras, are NP-hard. An obvious open question is to identify for which sets S, an S-factor is always guaranteed to exist; this amounts to resolving certain open problems in graph theory, even for some small specific S, and looks rather challenging.

More generally, the goal of this line of research is to obtain a complete classification of the computational complexity of $\mathrm{CSP}_2(\Gamma)$, analogous to Schaefer's Dichotomy Theorem. While an explicit classification may encounter difficult graph-theoretic questions, even for some specific Γ, it may well be possible to prove a general complexity dichotomy theorem, as we have done here, without having to resolve explicitly all the hard graph-theoretic questions.

One can observe that all the NP-hardness results of $\mathrm{CSP}_2(\Gamma)$, for the special case when Γ consists of symmetric relation(s), are established via the route of showing that Γ implements the Equality relation. This, in turn, allows to apply Schaefer's Dichotomy Theorem. One interesting open question is whether there exists a set Γ (consisting of not necessarily symmetric relations) that does not implement Equality, yet for which $\mathrm{CSP}_2(\Gamma)$ is NP-Hard. This would imply that Schaefer's Dichotomy does not cover all the cases of bounded read.

References

1 S. Akbari and M. Kano. $\{k, r-k\}$-Factors of r-Regular Graphs. *Graphs and Combinatorics*, 30(4):821–826, 2014. doi:10.1007/s00373-013-1324-x.

2 J. Akiyama and M. Kano. Factors and factorizations of graphs - a survey. *Journal of Graph Theory*, 9(1):1–42, 1985. doi:10.1002/jgt.3190090103.

3 E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: Refining Schaefer's theorem. *J. Comput. Syst. Sci.*, 75(4):245–254, 2009. doi:10.1016/j.jcss.2008.11.001.

4 C. Berge. *Graphs and Hypergraphs.* North-Holland mathematical library. Amsterdam, 1973.

5 A. Bernshteyn, O. Khormali, R. R. Martin, J. Rollin, D. Rorabaugh, S. Shan, and A. J. Uzzell. Regular colorings and factors of regular graphs. *CoRR*, abs/1603.09384, 2016. arXiv:1603.09384.

6 A. A. Bulatov. A Dichotomy Theorem for Nonuniform CSPs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.

7 V. Dalmau and D. K. Ford. Generalized Satisfability with Limited Occurrences per Variable: A Study through Delta-Matroid Parity. In *The 28th International Symposium Mathematical Foundations of Computer Science MFCS*, pages 358–367, 2003. doi:10.1007/978-3-540-45138-9_30.

8 Z. Dvorak and M. Kupec. On Planar Boolean CSP. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 432–443, 2015. doi:10.1007/978-3-662-47672-7_35.

9 J. Edmonds. Paths, Trees and Flowers. *CANADIAN JOURNAL OF MATHEMATICS*, pages 449–467, 1965.

10 T. Feder. Fanout limitations on constraint systems. *Theor. Comput. Sci.*, 255(1-2):281–293, 2001. doi:10.1016/S0304-3975(99)00288-1.

11 T. Feder and D. K. Ford. Classification of Bipartite Boolean Constraint Satisfaction through Delta-Matroid Intersection. *SIAM J. Discrete Math.*, 20(2):372–394, 2006. doi:10.1137/S0895480104445009.

12 T. Feder and M. Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

13 T. Gallai. On factorisation of graphs. *Acta Mathematica Academiae Scientiarum Hungarica*, 1(1):133–153, March 1950. doi:10.1007/BF02022560.

14 J. F. Geelen, S. Iwata, and K. Murota. The linear delta-matroid parity problem. *J. Comb. Theory, Ser. B*, 88(2):377–398, 2003. doi:10.1016/S0095-8956(03)00039-X.

15 G. Istrate. Looking for a Version of Schaefer"s Dichotomy Theorem When Each Variable Occurs at Most Twice. Technical report, University of Rochester, Rochester, NY, USA, 1997.

16 M. Kano and H. Matsuda. Partial Parity (g, f)-Factors and Subgraphs Covering Given Vertex Subsets. *Graphs and Combinatorics*, 17(3):501–509, 2001. doi:10.1007/PL00013412.

17 A. Kazda, V. Kolmogorov, and M. Rolínek. Even Delta-Matroids and the Complexity of Planar Boolean CSPs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 307–326, 2017. doi:10.1137/1.9781611974782.20.

18 L. Lovász. The factorization of graphs. II. *Acta Mathematica Academiae Scientiarum Hungarica*, 23(1):223–246, March 1972. doi:10.1007/BF01889919.

19 H. Lu, D. G. L. Wang, and Q. Yu. On the Existence of General Factors in Regular Graphs. *SIAM J. Discrete Math.*, 27(4):1862–1869, 2013. doi:10.1137/120895792.

20 J. Petersen. Die Theorie der regulären graphs. *Acta Mathematica*, 15:193–220, 1891. doi:10.1007/BF02392606.

21 M. D. Plummer. Graph factors and factorization: 1985–2003: A survey. *Discrete Mathematics*, 307(7-8):791–821, 2007.

22 T. J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978. doi:10.1145/800133.804350.

23 W.T. Tutte. The Subgraph Problem. In *Advances in Graph Theory*, volume 3 of *Annals of Discrete Mathematics*, pages 289–295. Elsevier, 1978. doi:10.1016/S0167-5060(08)70514-4.

24 D. Zhuk. A Proof of CSP Dichotomy Conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 331–342, 2017. doi:10.1109/FOCS.2017.38.

More on $\mathrm{AC}^0[\oplus]$ and Variants of the Majority Function

Nutan Limaye
Department of Computer Science and Engineering, IIT Bombay, India
nutan@cse.iitb.ac.in

Srikanth Srinivasan
Department of Mathematics, IIT Bombay, India
srikanth@math.iitb.ac.in

Utkarsh Tripathi
Department of Mathematics, IIT Bombay, India
utkarshtripathi.math@gmail.com

Abstract

In this paper we prove two results about $\mathrm{AC}^0[\oplus]$ circuits.

- We show that for $d(N) = o(\sqrt{\log N/\log\log N})$ and $N \leq s(N) \leq 2^{dN^{1/4d^2}}$ there is an explicit family of functions $\{f_N : \{0,1\}^N \to \{0,1\}\}$ such that
  - f_N has uniform AC^0 formulas of depth d and size at most s;
  - f_N does not have $\mathrm{AC}^0[\oplus]$ formulas of depth d and size s^{ε}, where ε is a fixed absolute constant.

  This gives a quantitative improvement on the recent result of Limaye, Srinivasan, Sreenivasaiah, Tripathi, and Venkitesh, (STOC, 2019), which proved a similar *Fixed-Depth Size-Hierarchy theorem* but for $d \ll \log\log N$ and $s \ll \exp(N^{1/2^{\Omega(d)}})$.

  As in the previous result, we use the *Coin Problem* to prove our hierarchy theorem. Our main technical result is the construction of uniform size-optimal formulas for solving the coin problem with improved sample complexity $(1/\delta)^{O(d)}$ (down from $(1/\delta)^{2^{O(d)}}$ in the previous result).
- In our second result, we show that randomness buys depth in the $\mathrm{AC}^0[\oplus]$ setting. Formally, we show that for any fixed constant $d \geq 2$, there is a family of Boolean functions that has polynomial-sized randomized uniform AC^0 circuits of depth d but no polynomial-sized (deterministic) $\mathrm{AC}^0[\oplus]$ circuits of depth d.

  Previously Viola (Computational Complexity, 2014) showed that an increase in depth (by at least 2) is essential to avoid superpolynomial blow-up while derandomizing randomized AC^0 circuits. We show that an increase in depth (by at least 1) is essential even for $\mathrm{AC}^0[\oplus]$.

  As in Viola's result, the separating examples are promise variants of the Majority function on N inputs that accept inputs of weight at least $N/2 + N/(\log N)^{d-1}$ and reject inputs of weight at most $N/2 - N/(\log N)^{d-1}$.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases $\mathrm{AC}^0[\oplus]$, Coin Problem, Promise Majority

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.22

Related Version A full version of the paper is available at `https://eccc.weizmann.ac.il/report/2019/133/`.

1 Introduction

This paper addresses questions in the field of *Boolean Circuit complexity,* where we study the complexity of compuational problems, modeled as sequences of Boolean functions $f_N : \{0,1\}^N \to \{0,1\}$, in the combinatorially defined Boolean circuit model (see, e.g. [5] for an introduction).

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 22; pp. 22:1–22:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Boolean circuit complexity is by now a classical research area in Computational complexity, with a large body of upper and lower bound results in many interesting circuit models. The questions we consider here are motivated by two of the most well-studied circuit models, namely AC^0 and AC$^0[\oplus]$. The circuit class AC^0 denotes the class of Boolean circuits of small-depth made up of AND, OR and NOT gates, while AC$^0[\oplus]$ denotes the circuit class that is also allowed the use of parity (addition modulo 2)[1] gates.[2]

Historically, AC^0 was among the first circuit classes to be studied and for which superpolynomial lower bounds were proved. Building on an influential line of work [2, 9, 24], Håstad [11] showed that any depth-d AC^0 circuit for the Parity function on N variables must have size $\exp(\Omega(N^{1/(d-1)}))$, hence proving an exponential lower bound for constant depths and superpolynomial lower bounds for all depths $d \ll \log N/\log\log N$. Researchers then considered the natural follow-up problem of proving lower bounds for AC$^0[\oplus]$. Soon after, Razborov [17] and Smolensky [21, 22] showed a lower bound of $\exp(\Omega(N^{1/2(d-1)}))$ for computing the Majority function on N inputs, again obtaining an exponential lower bound for constant depths and superpolynomial lower bounds for all depths $d \ll \log N/\log\log N$.

Thus, we have strong lower bounds for both classes AC^0 and AC$^0[\oplus]$. However, in many senses, AC$^0[\oplus]$ remains a much more mysterious class than AC^0. There are many questions that we have been successfully able to answer about AC^0 but whose answers still evade us in the AC$^0[\oplus]$ setting. This work is motivated by two such questions that we now describe.

Size Hierarchy Theorems. Size Hierarchy theorems are an analogue in the Boolean circuit complexity setting of the classical Time and Space hierarchy theorems for Turing Machines. Formally, the problem is to separate the power of circuits (from some class) of size s from that of circuits of size at most s^ε for some fixed $\varepsilon > 0$. As is usual in the setting of circuit complexity, we ask for *explicit* separations,[3] or equivalently, we ask that the separating sequence of functions be computed by a uniform family of circuits of size at most s.

The challenge here is to obtain explicit functions for which we can obtain *tight* (or near-tight) lower bounds, since we want the functions to have (uniform) circuits of size s but no circuits of size at most s^ε.

In the AC^0 setting, Håstad's theorem stated above immediately implies such a tight lower bound, since it is known (folklore) that the Parity function does have depth-d circuits of size $\exp(O(N^{1/d-1}))$ for every d. Varying the number of input variables to the Parity function suitably, this yields a Size Hierarchy theorem for the class of AC^0 circuits of depth d as long as $d \ll \log N/\log\log N$ and $s = \exp(o(N^{1/(d-1)}))$.

For AC$^0[\oplus]$, however, this is not as clear, as explicit *tight* lower bounds are harder to prove. In particular, the lower bounds of Razborov [17] and Smolensky [21, 22] for the Majority function (and other symmetric functions) are not tight; indeed, the exact complexity of these functions in AC$^0[\oplus]$ remains unknown [16]. In a recent result, the authors along with Sreenivasaiah and Venkitesh [14] were able to show a size hierarchy theorem for AC$^0[\oplus]$ *formulas*[4] for depths $d \ll \log\log N$ and size $s \ll \exp(N^{1/2^{\Omega(d)}})$. This is a weaker size

[1] Though we state our results only for AC$^0[\oplus]$, they extend in a straightforward way to AC$^0[p]$, where we are allowed gates that add modulo p, for any fixed prime p.

[2] The formal definitions of AC^0 and AC$^0[\oplus]$ only allow for polynomial-size circuits and constant depth. However, since some of our results apply to larger families of circuits, we will abuse notation and talk about AC^0 circuits of size $s(N)$ and depth $d(N)$ where s and d are growing functions of N.

[3] It is trivial to show a non-explicit separation by counting arguments.

[4] A formula is a circuit where the underlying undirected graph is a tree. For constant-depth, formulas and circuits are interchangeable with a polynomial blowup in depth. However, this is no longer true at superconstant depth [18, 19].

hierarchy theorem than the one that follows from Håstad's theorem for AC^0, both in terms of the size parameter as well as the depths until which it holds. In this paper, we build upon the ideas in [14] and prove the following result that is stronger in both parameters.

▶ **Theorem 1.** *The following holds for some absolute constant $\varepsilon > 0$. Let N be a growing parameter and $d = d(N), s = s(N)$ be functions of N with $d = o\left(\sqrt{\frac{\log N}{\log\log N}}\right)$ and $N \leq s \leq 2^{dN^{1/d^2}}$. Then there is a family of functions $\{f_N\}$ such that f_N has uniform AC^0 formulas of depth d and size at most s but no $\mathrm{AC}^0[\oplus]$ formulas of depth d and size at most s^{ε}.*

Randomized versus Deterministic circuits. The study of the relative power of randomized versus deterministic computation is an important theme in Computational complexity. In the setting of circuit complexity, it is known from a result of Adleman [1] that unbounded-depth polynomial-sized *randomized* circuits[5] are no more powerful than polynomial-sized deterministic circuits.

However, the situation is somewhat more intriguing in the bounded-depth setting. Ajtai and Ben-Or [3] showed that for any randomized depth-d AC^0 circuit of size at most s, there is deterministic AC^0 circuit of depth $d+2$ and size at most $\mathrm{poly}(s)$ that computes the same function; a similar result also follows for $\mathrm{AC}^0[\oplus]$ with the deterministic circuit having depth $d+3$. This begs the question: is this increase in depth necessary?

For AC^0 circuits of constant depth, Viola [23] gave an optimal answer to this question by showing that an increase of two in depth is necessary to avoid a superpolynomial blow-up in size. To the best of our knowledge, this problem has not been studied in the setting of $\mathrm{AC}^0[\oplus]$. In this paper, we show that an increase in depth (of at least one) is required even for $\mathrm{AC}^0[\oplus]$. More formally we prove the following theorem.

▶ **Theorem 2.** *Fix any constant $d \geq 2$. There is a family of Boolean functions that has polynomial-sized randomized uniform AC^0 circuits of depth d but no polynomial-sized (deterministic) $\mathrm{AC}^0[\oplus]$ circuits of depth d.*

Theorems 1 and 2 are proved in Sections 2 and 3 respectively. Many proofs are omitted for lack of space.

1.1 Proof Ideas

The proofs of both theorems are based on analyzing the complexity of Boolean functions that are closely related to the Majority function.

Size-Hierarchy Theorem. To prove the size hierarchy theorem for constant-depth $\mathrm{AC}^0[\oplus]$ formulas, [14] studied the $\mathrm{AC}^0[\oplus]$ complexity of the δ*-coin problem* [7], which is the problem of distinguishing between a coin that is either heads with probability $(1+\delta)/2$ or is heads with probability $(1-\delta)/2$, given a sequence of a large number of independent tosses of this coin. This problem has been studied in a variety of computational models [20, 7, 8, 10]. It is known [15, 4] that this problem can be solved by AC^0 formulas of depth d and size $\exp(O(d(1/\delta)^{1/(d-1)}))$ and further [15, 20, 14] that this upper bound is tight up to the constant in the exponent even for $\mathrm{AC}^0[\oplus]$ formulas of depth d. This gives a family of functions for which we have tight lower bounds for $\mathrm{AC}^0[\oplus]$ formulas.

[5] A *randomized Boolean circuit* for a Boolean function $f(x)$ is a Boolean circuit C that takes as input variables x and r such that for each setting of x and uniformly random r, $C(x) = f(x)$ with probability at least $3/4$.

Based on this, [14] noted that to prove $\mathrm{AC}^0[\oplus]$ size-hierarchy theorems for size $s(N)$ and depth $d(N)$, it suffices to construct a uniform sequence of formulas of size s and depth d solving the coin problem optimally (i.e. for δ such that $s = \exp(O(d(1/\delta)^{1/(d-1)})))$ using at most N samples. Before [14], all known size-optimal formula constructions for solving the δ-coin problem used $N = s = \exp(O(d(1/\delta)^{1/(d-1)}))$ many samples. The work of [14] brought the number of samples down to $N = (1/\delta)^{2^{O(d)}}$. Our main technical result here is an explicit size-optimal formula for solving the δ-coin problem using only $(1/\delta)^{O(d)}$ samples. Plugging this into the framework from [14], we immediately get the improved size-hierarchy theorem.

While the reason for this improvement is rather technical, we try to give a high-level outline here. It was shown by O'Donnell and Wimmer [15] and Amano [4] that the δ-coin problem is solved by read-once AC^0 formulas of depth d with gates of prescribed fan-ins. While the size s of these formulas is optimal, the number of samples is $N = s$, which is too big for our purposes. In [14], this number is brought down by distributing a smaller number of variables across the formula in a pseudorandom way (specifically using a Nisan-Wigderson design). The challenge now is to show that the formula still solves the δ-coin problem: the reason this is challenging is that various subformulas now share variables and hence the events that they accept or reject are no longer independent. However [14] note that *Janson's inequality* [13], a tool from probabilistic combinatorics, can be used to argue that if the variables are spread out in a suitably "random"-like fashion, then various subformulas at a certain depth may, for our intents and purposes, be treated as "nearly" independent.

This "distance" from independence is determined by a parameter Δ that goes into the statement of Janson's inequality, and hence let us call it the *Janson parameter*. In [14], this parameter was measured in a very brute-force way, forcing us to square the number of samples every time the depth of the formula increased by 1. This leads to a sample complexity of $(1/\delta)^{2^{O(d)}}$. Here, however, we give a different way of bounding the Janson parameter via a recursive analysis, which works as long as the number of variables grows by a factor of $(1/\delta)$ for each additional depth. This gives the improvement in our construction.

Randomized versus Deterministic circuits. For his separation of deterministic and randomized AC^0 circuits, Viola [23] used the *k-Promise-Majority* functions [6] which are Boolean functions that accept inputs with at least $N/2 + k$ many 1s and reject inputs with at most $N/2 - k$ many 0s. Building on work of [3, 15, 4], Viola [23] showed that for $k = N/(\log N)^{d-1}$, there are k-Promise-Majorities that have uniform polynomial-sized *randomized* depth-d AC^0 circuits. On the other hand, he also showed that the same problem has no deterministic circuit of depth d (and in fact even $d + 1$).

The challenge in proving such a lower bound is that if a Boolean function has a randomized circuit of depth d and size s, then it immediately follows that there is also a deterministic circuit of the same depth and size *approximating* the same Boolean function (i.e. computing it correctly on most inputs). In particular, the lower bound technique must be able to distinguish circuits that are computing the function exactly (since this is hard) from circuits that are merely approximating it (as this is easy). Viola overcomes this hurdle in the case of AC^0 with a clever argument for depth-3 circuits and an inductive use of the Håstad Switching lemma for higher depths. Neither of these techniques is available for $\mathrm{AC}^0[\oplus]$ circuits. In fact,

[6] These are called *Approximate Majorities* in a lot of the earlier literature, including in Viola's work. We avoid this name, since Approximate Majorities are also used for functions more closely related to the coin problem [15], and in our opinion, the name "Promise Majorities" better describes these functions.

the standard techniques for proving lower bounds against $\mathrm{AC}^0[\oplus]$ involve approximating the circuits to constant error using low-degree polynomials from $\mathbb{F}_2[x_1, \ldots, x_N]$. Note that this immediately runs into the obstacle mentioned above since we can then no longer distinguish between circuits that are exactly correct and those that are approximately correct.

The way we get around this argument is to use a recent result of Oliveira, Santhanam and the second author [16] where it is observed that the standard construction of approximating polynomials for $\mathrm{AC}^0[\oplus]$ actually gives polynomials that approximate the given circuit C to very small error on either the zero or the one inputs of C. They are able to use this to improve known $\mathrm{AC}^0[\oplus]$ lower bounds for the Majority function. Our main observation is that this stronger lower bound is actually able to distinguish between circuits that approximate the Majority function to constant error (say from [15, 4]) and those that compute it exactly, thus overcoming the barrier we mentioned above. We then note that their proof can also be made to work for k-Promise-Majorities. This yields the separation.

2 Size hierarchy theorem for $\mathrm{AC}^0[\oplus]$

▶ **Definition 3** (The δ-Coin Problem). *Let $\delta \in (0, 1)$ be a parameter. Given an $N \in \mathbb{N}$, we define the probability distributions $\mu^N_{\delta,0}$ and $\mu^N_{\delta,1}$ to be the product distributions where each bit is set to 1 with probability $(1-\delta)/2$ and $(1+\delta)/2$ respectively. We omit the δ in the subscript and N in the superscript when these are clear from context.*

Given a function $g : \{0, 1\}^N \to \{0, 1\}$, we say that g solves the δ-coin problem *if*

$$\Pr_{\boldsymbol{x}\sim\mu_0^N}[g(\boldsymbol{x}) = 1] \leq 0.1 \text{ and } \Pr_{\boldsymbol{x}\sim\mu_1^N}[g(\boldsymbol{x}) = 1] \geq 0.9. \tag{1}$$

We say that the sample complexity *of g is N.*

Parameters. Let m, d be growing parameters such that $d = o(m/\log m)$. Let $1/\delta = (m \ln 2)^{d-1}/C_1$, where C_1 is a fixed large constant, to be specified below. Let $M = \lceil m{\cdot}2^m{\cdot}\ln 2\rceil$ and let $M_1 = 2^m$.

▶ **Theorem 4.** *For large enough absolute constant C_1, the following holds. For parameters m, δ, d as above and for $d \geq 2$, there is an explicit depth-d AC^0 formula of size $\exp(O(dm))$ $= \exp(O(d(1/\delta)^{1/d-1}))$ and sample complexity $(1/\delta)^{d+4}$ that solves the δ-coin problem.*

We first show how Theorem 4 implies Theorem 1 stated in the introduction.

Proof of Theorem 1. We use Theorem 4 for a suitable choice of parameters to define the explicit function.

Let $m = \lfloor(\alpha \log s)/d\rfloor$ for some absolute constant $\alpha < 1$ that we fix below. It can be checked that as $s \geq N$ and $d = o(\sqrt{\log N/\log\log N})$, we have $d = o(m/\log m)$. Define δ as above and note that $(1/\delta)^{d+4} \leq m^{2d^2} \leq ((\log s)/d)^{2d^2} \leq N$, where the final inequality uses the given upper bounds on d and s.

We set f_N to be the Boolean function computed by the formula F_d constructed above on the first $(1/\delta)^{d+4}$ of the N input variables. By Theorem 4, the size of F_d is $\exp(O(dm)) \leq s$ for a small enough absolute constant α and F_d solves the δ-coin problem. Moreover, it was shown in [14] that any depth-d $\mathrm{AC}^0[\oplus]$ formula solving the δ-coin problem must have size $\exp(\Omega(d(1/\delta)^{1/(d-1)})) = \exp(\Omega(md)) = s^\varepsilon$ for some absolute constant $\varepsilon > 0$. This proves the theorem. ◀

2.1 Proof of Theorem 4

In this section we give the construction of the explicit formula solving the δ-coin problem and prove Theorem 4. There exist integers Q, D, such that[7] Q is a prime power, $M \leq Q^D \leq 2M$ and $(m^4/\delta) \leq Q \leq (2m^4/\delta)$. Let $\mathbb{F}$ be a finite field with Q elements and $A \subseteq \mathbb{F}$ be a set of size m. Let $\mathcal{P}_D$ be the lexicographically first M univariate polynomials over $\mathbb{F}$ of degree strictly less than D. Similarly, let $\mathcal{P}'_D$ be the lexicographically first M_1 univariate polynomials over $\mathbb{F}$ of degree less than D.

We now describe the construction of our formula. The variables in the formula correspond to the points in the set $A \times \mathbb{F}^{d-1}$. i.e. for each $(a, c_1, \dots, c_{d-1}) \in A \times \mathbb{F}^{d-1}$, we have a variable $x(a, c_1, \dots, c_{d-1})$. We thus have $m \cdot Q^{d-1}$ many variables, denoted by N.

For each $i \in [d-1]$ and $\bar{P} = (P_i, \dots, P_{d-1}) \in \mathcal{P}_D^{d-i}$, define a depth-$i$ formula $\mathcal{C}_{(P_i,\dots,P_{d-1})}$ inductively as follows.

$$\mathcal{C}_{(P_1,\dots,P_{d-1})} = \bigwedge_{a \in A} x(a, P_1(a), \dots, P_{d-1}(a)),$$

$$\mathcal{C}_{(P_2,\dots,P_{d-1})} = \bigvee_{R_1 \in \mathcal{P}_D} \mathcal{C}_{(R_1,P_2,\dots,P_{d-1})}, \qquad \mathcal{C}_{(P_3,\dots,P_{d-1})} = \bigwedge_{R_2 \in \mathcal{P}_D} \mathcal{C}_{(R_2,P_3,\dots,P_{d-1})}$$

and so on, with the gates alternately repeating between **AND** and **OR** untill depth $d-1$. Finally, $\mathcal{C}_{(\emptyset)}$ is the output of the formula. If the depth of the formula is odd then $\mathcal{C}_{(\emptyset)}$ is equal to $\bigwedge_{R \in \mathcal{P}'_D} \mathcal{C}_{(R)}$ otherwise it is equal to $\bigwedge_{R \in \mathcal{P}'_D} \mathcal{C}_{(R)}$. This finishes the description of our formula. We use $F_d = \mathcal{C}_{(\emptyset)}$ to denote this formula.

Analysis of the construction

Here we present the details regarding the analysis of our construction presented above, which will be used to prove Theorem 4. We will start with some definitions, notations and some useful inequalities.

▶ **Definition 5.** *For $1 \leq i \leq d-1$, we define the following terms.*

1. *For $\bar{P} = (P_i, \dots, P_{d-1}) \in \mathcal{P}_D^{d-i}$ and $b \in \{0,1\}$, let*

$$Acc_{\bar{P},b} := \Pr_{\mu_b}[\mathcal{C}_{(P_i,\dots,P_{d-1})} \text{ accepts}] \text{ and } Rej_{\bar{P},b} := \Pr_{\mu_b}[\mathcal{C}_{(P_i,\dots,P_{d-1})} \text{ rejects}].$$

   Let $q_{\bar{P},b} = Acc_{\bar{P},b}$ if i is odd and $Rej_{\bar{P},b}$ if i is even.
2. *For $\bar{P} = (P_i, \dots, P_{d-1}), \bar{P}' = (P'_i, \dots, P'_{d-1}) \in \mathcal{P}_D^{d-i}$, we say that $\bar{P} \sim \bar{P}'$ when $\mathcal{C}_{\bar{P}}$ and $\mathcal{C}_{\bar{P}'}$ are distinct gates which share a common input variable.*
3. *Fix any $i \in [d-1]$. For $\bar{P} = (P_{i+1}, \dots, P_{d-1}), \bar{P}' = (P'_{i+1}, \dots, P'_{d-1}) \in \mathcal{P}_D^{d-i-1}$, $b \in \{0,1\}$,*

$$\Delta_{\bar{P},\bar{P}',b} = \begin{cases} \displaystyle\sum_{\substack{R_i, R'_i \in \mathcal{P}_D \\ (R_i,\bar{P}) \sim (R'_i,\bar{P}')}} \mathbf{Pr}_{\mu_b}[\mathcal{C}_{(R_i,\bar{P})} = 0 \textbf{ AND } \mathcal{C}_{(R'_i,\bar{P}')} = 0] & \text{if } \mathcal{C}_{\bar{P}} \text{ and } \mathcal{C}_{\bar{P}'} \text{ are AND gates} \\ \displaystyle\sum_{\substack{R_i, R'_i \in \mathcal{P}_D \\ (R_i,\bar{P}) \sim (R'_i,\bar{P}')}} \mathbf{Pr}_{\mu_b}[\mathcal{C}_{(R_i,\bar{P})} = 1 \textbf{ AND } \mathcal{C}_{(R'_i,\bar{P}')} = 1] & \text{if } \mathcal{C}_{\bar{P}} \text{ and } \mathcal{C}_{\bar{P}'} \text{ are OR gates} \end{cases}$$

[7] Using number-theoretic facts about the density of primes [6] (see for instance [12]), such Q, D can be found in polynomial time.

A useful tool in our analysis of the circuit is Janson's inequality stated here in the language of Boolean circuits.

▶ **Theorem 6** (Janson's inequality). *Let $\mathcal{C}_1, \ldots, \mathcal{C}_M$ be any monotone Boolean circuits over inputs $x_1, \ldots, x_n$ and let $\mathcal{C}$ denote $\bigvee_{i\in[M]} \mathcal{C}_i$. For each distinct $i, j \in [M]$, we use $i \sim j$ to denote the fact that $\mathcal{C}_i$ and $\mathcal{C}_j$ share a common variable. Assume each x_j ($j \in [M]$) is chosen independently to be 1 with probability $p_j \in [0,1]$, and that under this distribution, we have $\mathbf{max}_{i\in[M]}\{\mathbf{Pr}_x[\mathcal{C}_i(x) = 1]\} \leq 1/2$. Then we have*

$$\prod_{i\in[M]} \mathbf{Pr}_x[\mathcal{C}_i(x) = 0] \;\leq\; \mathbf{Pr}_x[\mathcal{C}(x) = 0] \;\leq\; \left(\prod_{i\in[M]} \mathbf{Pr}_x[\mathcal{C}_i(x) = 0]\right) \cdot \exp(\Delta) \tag{2}$$

where $\Delta := \sum_{i\sim j} \mathbf{Pr}_x[(\mathcal{C}_i(x) = 1) \wedge (\mathcal{C}_j(x) = 1)]$.

Throughout, we use $\log(\cdot)$ to denote logarithm to the base 2 and $\ln(\cdot)$ for the natural logarithm. We use $\exp(x)$ to denote e^x.

▶ **Fact 7.** *Assume that $x \in [-1/2, 1/2]$. Then we have the following chain of inequalities.*

$$\exp(x - (|x|/2)) \underset{\text{(a)}}{\leq} \exp(x - x^2) \underset{\text{(b)}}{\leq} 1 + x \underset{\text{(c)}}{\leq} \exp(x) \underset{\text{(d)}}{\leq} 1 + x + x^2 \underset{\text{(e)}}{\leq} 1 + x + (|x|/2) \tag{3}$$

We define a few parameters which will be useful in the main technical lemma that helps in proving Theorem 4.

For $i \in [d-1]$, let $\alpha_i = m^i \cdot (\ln 2)^{i-1} \cdot \delta$. Also define $\beta_1 = 2\alpha_1$ and $\beta_i = \beta_{i-1} + 2\alpha_i + \frac{2}{m^i(\ln 2)^{i-1}}$ for $2 \leq i \leq d-2$.

▶ **Observation 8.** *For all $i \in [d-2]$, $\alpha_i, \beta_i \leq O(1/m)$. Also, $\alpha_{d-1} = \Theta(C_1) = \Theta(1)$. Finally, for $i \in [d-2]$, using Fact 7 above, we get $\exp(-\beta_{i-1}) - \exp(-\beta_i) \geq \alpha_i/2$.*

▶ **Lemma 9.** *Assume $d \geq 3$ and $q_{\bar{P},b}$ and formula $\mathcal{C}_{(\emptyset)}$ defined as before. We have the following properties.*

1. *For $b \in \{0,1\}$, $i \in [d-2]$ such that $i \equiv b \pmod 2$,*

$$\frac{1}{2^m} \cdot (1 + \alpha_i \exp(-\beta_i)) \leq q_{\bar{P},b} \leq \frac{1}{2^m} \cdot (1 + \alpha_i \exp(\beta_i))$$
$$\frac{1}{2^m} \cdot (1 - \alpha_i \exp(\beta_i)) \leq q_{\bar{P},(1-b)} \leq \frac{1}{2^m} \cdot (1 - \alpha_i \exp(-\beta_i))$$

2. *Say $d - 1 \equiv b \pmod 2$. Then*

$$q_{\bar{P},b} \geq \frac{1}{2^m} \cdot \exp(\alpha_{d-1}/4) \text{ and } q_{\bar{P},1-b} \leq \frac{1}{2^m} \cdot \exp(-\alpha_{d-1}/4)$$

3. *For all $i \in [d-1]$, $b \in \{0,1\}$ and $\bar{P}, \bar{P}' \in \mathcal{P}_D^{d-i-1}$, $\Delta_{\bar{P},\bar{P}',b} < \delta$.*

Assuming that the above lemma holds for now, we will prove Theorem 4.

Proof of Theorem 4. We start by bounding the size of $F_d = \mathcal{C}_{(\emptyset)}$. As per our construction, the gates at level 1 are **AND** gates with fan-in m each. For all $2 \leq i \leq d-1$, the fan-in of each gate on level i is $M = \lceil m \cdot 2^m \cdot \ln 2\rceil$ and the top fan-in is $M_1 = 2^m$. Therefore, the total number of gates in the formula is $m \cdot M^{d-2} \cdot M_1$. We can trivially bound this by $M^d = O(m^d 2^{dm})$. As $d = o(m/\log m)$, we get that the size is bounded by $\exp(O(dm))$. Recall that $1/\delta = (m \ln 2)^{d-1}/C_1$, where C_1 is an appropriately chosen constant. Hence $\exp(O(dm)) = \exp(O(d(1/\delta)^{1/(d-1)}))$.

We will now bound the number of variables N used by the formula. As mentioned above, $N = m \cdot Q^{d-1}$. As Q is chosen such that $Q = \Theta(m^4/\delta)$, there exists a constant C' such that $N \leq m \cdot (C'm^4/\delta)^{d-1}$.

$$\begin{aligned} N &\leq m \cdot (C'm^4/\delta)^{d-1} \leq (1/\delta)^{d-1} \cdot (m^{d-1})^4 \cdot m \cdot C'^{d-1} \\ &\leq (\frac{1}{\delta})^{d-1} \cdot (\frac{1}{\delta})^4 \cdot m \cdot (C'')^{d-1} \quad \text{(for some constant } C'' \text{ as } 1/\delta = (m\ln 2)^{d-1}/C_1) \\ &\leq (\frac{1}{\delta})^{d+3} \cdot \frac{1}{\delta} = (\frac{1}{\delta})^{d+4} \end{aligned}$$

Finally, we will show that the formula solves the δ-coin problem. Let us assume that d is even. In that case, the output gate $\mathcal{C}_{(\emptyset)}$ is an **OR** gate. (When it is an **AND** gate, the analysis is very similar.) We bound the probabilities $\Pr_{a\in\mu_0}[F_d(a) = 1]$ and $\Pr_{a\in\mu_1}[F_d(a) = 0]$ by 1/10 each.

$$\begin{aligned} \Pr_{a\in\mu_0}[F_d(a) = 1] &\leq \sum_{R\in\mathcal{P}'_D} \Pr_{a\in\mu_0}[\mathcal{C}_{(R)}(a) = 1] && \text{Using a Union bound} \\ &\leq 2^m \cdot \frac{1}{2^m} \cdot \exp(-\alpha_{d-1}/4) && |\mathcal{P}'_D| = 2^m, \text{ using Lemma 9, (2)} \\ &\leq \exp(-\Omega(C_1)) && \text{Using the value of } \alpha_{d-1} \\ &\leq 1/10. && \text{for large enough } C_1 \end{aligned}$$

$$\begin{aligned} \Pr_{a\in\mu_1}[F_d(a) = 0] &\leq \prod_{\bar{P}\in\mathcal{P}'_D} \Pr_{a\in\mu_1}[\mathcal{C}_{(\bar{P})}(a) = 0] \cdot \exp(\delta) && \text{Using Janson's inequality} \\ & && \text{and Lemma 9, (3)} \\ &\leq \prod_{\bar{P}\in\mathcal{P}'_D} (1 - \Pr_{a\in\mu_1}[\mathcal{C}_{(\bar{P})}(a) = 1]) \cdot \exp(\delta) \\ &\leq (1 - \frac{1}{2^m} \cdot \exp(\alpha_{d-1}/4))^{2^m} \cdot \exp(\delta) && |\mathcal{P}'_D| = 2^m, \text{ using Lemma 9, (2)} \\ &\leq \exp\left(-\frac{2^m}{2^m} \cdot \exp(\alpha_{d-1}/4)\right) \cdot 2 && \text{As } \exp(\delta) \leq 2 \\ &\leq 1/10. && \text{Using the value of } \alpha_{d-1} \\ & && \text{and for large enough } C_1 \end{aligned}$$

This finishes the proof of Theorem 4 assuming Lemma 9. ◀

We now give the proof of Lemma 9. The proof is by induction on the depth of the circuit.

Proof of Lemma 9. The lemma has three parts. As mentioned above, we proceed by induction on the depth.

Base case ($i = 1$): Here let us first assume that we are working with μ_1^N. We start with part (1). We wish to bound $q_{\bar{P},1}$. From the construction of our formula, we know that the formula has **AND** gates at layer 1 and the inputs to these are distinct variables and hence independent. Therefore, $q_{\bar{P},1} = \left(\frac{1+\delta}{2}\right)^m$. We will upper and lower bound this quantity.

$$\left(\frac{1+\delta}{2}\right)^m \geq \frac{1}{2^m} \cdot (1 + \delta m) \geq \frac{1}{2^m} \cdot (1 + \alpha_1 \cdot \exp(-\beta_1)) \quad (\text{As } \alpha_1 = \delta m, \beta_1 > 0)$$

$$\begin{aligned}
\left(\frac{1+\delta}{2}\right)^m &= \frac{1}{2^m}\cdot(1+\delta)^m \leq \frac{1}{2^m}\cdot\exp(\delta m) && \text{Fact 7 (c)}\\
&\leq \frac{1}{2^m}\cdot(1+\delta m+(\delta m)^2) && \text{Fact 7 (d)}\\
&\leq \frac{1}{2^m}\cdot(1+\delta m\cdot\exp(2\delta m)) && \text{Fact 7 (c)}\\
&= \frac{1}{2^m}\cdot(1+\alpha_1\cdot\exp(\beta_1)) && \text{As } \alpha_1=\delta m, \beta_1 = 2\alpha_1
\end{aligned}$$

In the case of μ_0^N, we get $q_{\bar{P},0} = \left(\frac{1-\delta}{2}\right)^m$ and a very similar computation can be used to upper and lower bound this quantity.

There is nothing to prove for part (2) in the base case. We now prove the base case for part (3). Let $\bar{P} = (P_2,\ldots,P_{d-1}), \bar{P}' = (P'_2,\ldots,P'_{d-1}) \in \mathcal{P}_D^{d-2}$. We will analyse $\Delta_{\bar{P},\bar{P}',1}$ here. The analysis for $\Delta_{\bar{P},\bar{P}',0}$ is very similar. Let λ denote $(1+\delta)/2$. For a formula F, let $\mathbf{Var}(F)$ denote the set of variables appearing in it.

$$\begin{aligned}
\Delta_{\bar{P},\bar{P}',1} &= \sum_{\substack{R,R'\in\mathcal{P}_D\\(R,\bar{P})\sim(R',\bar{P}')}} \Pr_{\mu_1}[C_{(R,\bar{P})}=1 \textbf{ AND } C_{(R',\bar{P}')}=1]\\
&= \sum_{\substack{R,R'\\(R,\bar{P})\sim(R',\bar{P}')}} \lambda^{|\mathbf{Var}(C_{(R,\bar{P})})\cup\mathbf{Var}(C_{(R',\bar{P}')})|}\\
&= \lambda^{2m}\cdot \sum_{\substack{R,R'\\(R,\bar{P})\sim(R',\bar{P}')}} \left(\frac{1}{\lambda}\right)^{|\mathbf{Var}(C_{(R,\bar{P})})\cap\mathbf{Var}(C_{(R',\bar{P}')})|}. \qquad (4)
\end{aligned}$$

To bound the above term, we use the following technical claim (proof omitted).

▷ Claim 10. Fix any $i \leq d-1$ and any $\bar{P},\bar{P}' \in \mathcal{P}_D^{d-i-1}$, we have

$$\sum_{\substack{R,R'\\(R,\bar{P})\sim(R',\bar{P}')}} 3^{|\mathbf{Var}(C_{(R,\bar{P})})\cap\mathbf{Var}(C_{(R',\bar{P}')})|} \leq Q^{2D}\cdot O\left(\frac{m}{Q}\right).$$

Using Claim 10 and (4), we can immediately bound $\Delta_{\bar{P},\bar{P}',1}$ as follows.

$$\Delta_{\bar{P},\bar{P}',1} \leq \lambda^{2m}\cdot \sum_{\substack{R,R'\\(R,\bar{P})\sim(R',\bar{P}')}} 3^{|\mathbf{Var}(C_{(R,\bar{P})})\cap\mathbf{Var}(C_{(R',\bar{P}')})|} \leq \lambda^{2m}Q^{2D}\cdot O\left(\frac{m}{Q}\right) \leq O\left(\frac{m^3}{Q}\right) < \delta$$

where we have used the fact that $\lambda \geq 1/3$, $Q^D = O(M^2) = O(m^2 2^{2m})$, and $Q = \Theta(m^4/\delta)$. This concludes the bound on $\Delta_{\bar{P},\bar{P}',1}$ and hence concludes the proof of the base case.

Inductive case: The proof of parts (1) and (2) are similar to the base case and hence omitted.

Finally, we prove the inductive statement about $\Delta_{\bar{P},\bar{P}',1}$ in the case that i is odd. Fix any $\bar{P},\bar{P}' \in \mathcal{P}_D^{d-i-1}$ (in the case that $i = d-1$, we will have $\bar{P} = \bar{P}' = (\emptyset)$). The computation goes as follows. *The crucial steps are the second equality and first inequality, where we interpret each term in the sum as the probability that a depth $i-1$ circuit takes the value* 0*, which is bounded using Janson's inequality and the induction hypothesis.*

$$\begin{aligned}
\Delta_{\bar{P},\bar{P}',1} &= \sum_{\substack{R,R' \\ (R,\bar{P})\sim(R',\bar{P}')}} \Pr_{\mu_1^N}[C_{(R,\bar{P})} = 0 \textbf{ AND } C_{(R',\bar{P}')} = 0] \\
&= \sum_{\substack{R,R' \\ (R,\bar{P})\sim(R',\bar{P}')}} \Pr_{\mu_1^N}[\bigvee_S C_{(S,R,\bar{P})} \vee \bigvee_{S'} C_{(S',R',\bar{P}')} = 0] \\
&\leq \sum_{\substack{R,R' \\ (R,\bar{P})\sim(R',\bar{P}')}} \prod_S \Pr[C_{(S,R,\bar{P})} = 0] \cdot \prod_{S'} \Pr[C_{(S',R',\bar{P}')} = 0] \cdot \exp(4\delta) \\
&\leq \exp(4\delta) \cdot \sum_{\substack{R,R' \\ (R,\bar{P})\sim(R',\bar{P}')}} \left(1 - \frac{(1-2\alpha_{i-1})}{2^m}\right)^{2M} = \exp(4\delta)\left(1 - \frac{(1-2\alpha_{i-1})}{2^m}\right)^{2M} \cdot \sum_{\substack{R,R' \\ (R,\bar{P})\sim(R',\bar{P}')}} 1 \\
&\leq 2\exp(-2m\ln 2 + O(\alpha_i)) \cdot \sum_{\substack{R,R' \\ (R,\bar{P})\sim(R',\bar{P}')}} 1 = O\left(\frac{1}{2^{2m}}\right) \cdot \sum_{\substack{R,R' \\ (R,\bar{P})\sim(R',\bar{P}')}} 1
\end{aligned}$$

where the first inequality is just Janson's inequality applied to the formula $\bigvee_S C_{S,R,\bar{P}} \vee \bigvee_{S'} C_{S',R',\bar{P}'}$; the second inequality follows from the induction hypothesis applied to level $i - 1 \leq d - 2$ (we have used a slightly weaker bound that is applicable also to other cases such as when $b = 0$); and the last inequality follows from our choice of M and the fact that $\alpha_i = \alpha_{i-1} \cdot (m \ln 2)$. The sum in the final term may be bounded by $Q^{2D} \cdot O(m/Q)$ by Claim 10. We thus get

$$\Delta_{\bar{P},\bar{P}',1} \leq O\left(\frac{Q^{2D}}{2^m}\right) \cdot \frac{m}{Q} = O\left(\frac{M^2}{2^m}\right) \cdot \frac{m}{Q} \leq \frac{O(m^3)}{Q} < \delta$$

as $Q \geq m^4/\delta$. This finishes the analysis of $\Delta_{\bar{P},\bar{P}',1}$. ◀

3 Randomized vs. Deterministic AC⁰[⊕] circuits

For $a \in \{0,1\}^n$, let $|a|$ denote the Hamming weight of a, i.e. the number of 1s in a.

▶ **Definition 11.** *Let $k, \ell \leq n/2$. The Promise Majority problem, $\mathsf{PrMaj}^n_{k,\ell}$, is a promise problem of distiguishing n-bit strings of Hamming weight less than $n/2 - k$ from those with Hamming weight more than $n/2 + \ell$. Formally,*

$$\mathsf{PrMaj}^n_{k,\ell}(a) = \begin{cases} 0 & \textit{if } |a| < (\frac{n}{2} - k) \\ 1 & \textit{if } |a| \geq (\frac{n}{2} + \ell) \end{cases}$$

If the length of the input is clear from the context then we drop the superscript n. If $k = 0$ then we denote $\mathsf{PrMaj}_{0,\ell}$ by $\mathsf{LowPrMaj}_\ell$. Similarly, $\ell = 0$ then we denote $\mathsf{PrMaj}_{k,0}$ by $\mathsf{UpPrMaj}_k$. When both k, ℓ are zero, $\mathsf{PrMaj}_{0,0}$ is the Majority function. If $k = \ell$ then we use PrMaj_k to denote $\mathsf{PrMaj}_{k,k}$.

Let $\mathsf{Yes}^n_\ell, \mathsf{No}^n_k$ denote the yes and no instances of $\mathsf{PrMaj}^n_{k,\ell}$. That is, $\mathsf{Yes}^n_\ell = \{a \in \{0,1\}^n \mid |a| \geq n/2 + \ell\}$ and $\mathsf{No}^n_k = \{a \in \{0,1\}^n \mid |a| < n/2 - k\}$. In [23], the following theorem was proved.

▶ **Theorem 12** (Theorem 1.2 [23]). *For any $d \geq 2$ and $k(N) = \Omega(N/(\log N)^{d-1})$, there is a uniform family of randomized AC^0 circuits of depth d and $\mathrm{poly}(N)$ size computing $\mathsf{PrMaj}^N_{k(N)}$.*

Here, we prove the following theorem.

▶ **Theorem 13.** *For any $d \geq 2$, say $\mathcal{C}$ is a (deterministic) $\mathrm{AC}^0[\oplus]$ circuit of depth d computing $\mathsf{PrMaj}^N_{N/2\cdot(\log N)^{d-1}}$, then $\mathcal{C}$ must have size $N^{\omega(1)}$.*

It is easy to see that using Theorem 12 and Theorem 13, we immediately get Theorem 2. In order to prove Theorem 13 we need the following claim. This is our main technical claim.

▷ Claim 14. Let $n \in \mathbb{N}$ and let $k = \Theta(n/(\log n)^c)$. Let $p \in \mathbb{F}[x_1, \ldots, x_n]$ be a (deterministic) polynomial such that it satisfies one of the following two conditions

$$\text{either} \quad \Pr_{a \in \mathsf{No}^n_k}[p(a) = 1] \leq 1/n, \qquad \Pr_{a \in \mathsf{Yes}^n_0}[p(a) = 0] \leq 1/10 \tag{5}$$

$$\text{or} \quad \Pr_{a \in \mathsf{No}^n_0}[p(a) = 1] \leq 1/10 \qquad \Pr_{a \in \mathsf{Yes}^n_k}[p(a) = 0] \leq 1/n \tag{6}$$

Then $\deg(p) = \Omega(\log^{c+1} n)$.

Proof of Theorem 13 using Claim 14. We will first show that Theorem 13 follows from the above claim. We will do this using the following two step argument.

(I) Let us assume for now that $\mathcal{C}$ is a circuit of size s and depth d with either OR gate or $\oplus$ gate as its output gate. Let us call the output gate G_{out}. We will show that if $\mathcal{C}$ computes PrMaj^N_k then we have a circuit $\mathcal{C}'$ of size s, depth d and with output gate G_{out}, such that it computes $\mathsf{UpPrMaj}^n_{2k}$, where $n = N - 2k$.[8]

(II) We will then show that any depth d circuit with OR or $\oplus$ output gate computing $\mathsf{UpPrMaj}^n_{2k}$ must have size $n^{\omega(1)}$.

As we will invoke this for $k = N/2(\log N)^c$, which is $o(N)$, an $n^{\omega(1)}$ lower bound on $\mathsf{UpPrMaj}^n_{2k}$ will imply a $N^{\omega(1)}$ lower bound on PrMaj^N_k, thereby proving the theorem.

Here, (I) can be shown by simply fixing some of the input bits to the constant 1. Specifically, let us set $2k$ bits out of the N bits to 1s. Let $n = N - 2k$. It is easy to see that if $x \in \{0,1\}^n$ has Hamming weight at least $n/2$, then in fact $y = x \cdot 1^{2k}$ has $N/2 + k$ many 1s. Similarly, if $x \in \{0,1\}^n$ has Hamming weight at most $n/2 - 2k$ then the Hamming weight of $y = x \cdot 1^{2k}$ is at most $N/2 - k$.

To show (II) requires a little more work. In particular, to show (II), we use a result from [16] about degree of polynomials approximating $\mathrm{AC}^0[\oplus]$ circuits. To state their result, we will introduce some notation.

▶ **Definition 15.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. For any parameters $\varepsilon_0, \varepsilon_1$, $(\varepsilon_0, \varepsilon_1)$-error probabilistic polynomial for f is a random multilinear polynomial P chosen from $\mathbb{F}_2[x_1, \ldots, x_n]$, such that for any $b \in \{0,1\}$ and any $a \in f^{-1}(b)$, $\Pr[P(a) \neq f(a)] \leq \varepsilon_b$.*

A probabilistic polynomial is said to have degree at most d if the underlying distribution is supported on monomials of degree at most d.

We define the $(\varepsilon_0, \varepsilon_1)$-error probabilistic polynomial degree of a Boolean function f, denoted as $\mathsf{pdeg}_{\varepsilon_0,\varepsilon_1}(f)$, to be the smallest d such that there is an $(\varepsilon_0, \varepsilon_1)$-error probabilistic polynomial of degree d for f.

[8] As PrMaj is a self-dual function and $\mathsf{UpPrMaj}$ and $\mathsf{LowPrMaj}$ are duals of each other, we can assume that the output gate of $\mathcal{C}$ is OR or $\oplus$ without loss of generality.

▶ **Lemma 16** (Corollary 15, [16]). *Let C be a size s, depth d circuit with OR or $\oplus$ as its output gate. Then there is a probabilistic polynomial $\overline{p}$ approximating C such that* $\mathsf{pdeg}_{1/n^2,1/100}(\overline{p})$ *is at most $O(\log s)^{d-1}$.*

▶ Remark 17. Let $\mathcal{C}$ be a circuit of size s and depth d (with any output gate). It is known that if $\overline{p}$ is a probabilistic polynomial for $\mathcal{C}$ such that $\varepsilon_0 = \varepsilon_1 = 1/s^{O(1)}$, then $\mathsf{pdeg}_{1/s^{O(1)},1/s^{O(1)}}(\overline{p})$ is $O(\log s)^d$. The above lemma says that if we need only constant error on one of sides, i.e. say if either ε_0 or ε_1 is $\Omega(1)$, then we can get a better degree upper bound. Instead of having d in the exponent, we get $d-1$ in the exponent. This is crucial.

Note that, if the output gate of $\mathcal{C}$ is OR (AND) then we can ensure that $\varepsilon_0 = 1/n^2$ ($\varepsilon_1 = 1/n^2$, resp.). If it is a $\oplus$ gate, then either can be ensured.

Suppose there is an AC$^0[\oplus]$ circuit $\mathcal{C}$ of size $s = n^t$ and depth d with top gate OR or $\oplus$ and computing $\mathsf{UpPrMaj}_{2k}$.

Applying Lemma 16 and by standard averaging arguments we can show that there is a fixed polynomial $P \in F[X]$ that satisfies conditions (5) for $c = d-1$ and has the same degree as the degree of $\overline{p}$. Therefore on the one hand, we know that $\deg(P)$ is less than or equal to $O(t \log n)^{d-1}$, while on the other hand using Claim 14 we get that $\deg(P)$ is at least $\Omega(\log n)^d$. (As $N/(\log N)^c = \Theta(n/(\log n)^c)$, Claim 14 is applicable.) Thus, $O(t \log n)^{d-1} \geq \Omega(\log n)^d$ and hence we get $t \geq \Omega(\log n)^{1/d-1}$. Therefore we get (II). This finishes the proof of Theorem 13. ◀

We now proceed with the proof of Claim 14. We will use the following fact in the proof of Claim 14.

▶ **Fact 18.** *Say $R \in \mathbb{F}[X]$ is a non-zero polynomial that vanishes on No_k^n, then degree of R is at least $n/2 - k$.*

Proof of Claim 14. We will show that if a deterministic polynomial $p \in \mathbb{F}[X]$ satisfies condition (5), then it has degree $C \cdot \log^{c+1} n$ for some constant C. The proof for the lower bound on the degree of p assuming condition (6) is similar. For simplicity we will work out the proof when $k = n/(\log n)^c$. The proof is similar when $k = \Theta(n/(\log n)^c)$.

Let us use D to denote $C \cdot \log^{c+1} n$. Consider a polynomial p satisfying condition (5). Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be error sets of this polynomial on no and yes instances respectively, i.e. $\mathcal{E}_0 = \{a \in \mathsf{No}^n_{n/(\log n)^c} \mid p(a) = 1\}$ and $\mathcal{E}_1 = \{a \in \mathsf{Yes}^n_0 \mid p(a) = 0\}$. From condition (5) we have a bound on the cardinalities of $\mathcal{E}_0, \mathcal{E}_1$.

We will first observe that in order to prove the claim, it suffices to show the existence of a polynomial $Q \in \mathbb{F}[X]$ with the following three properties.

(a) $Q(a) = 0$ for all $a \in \mathcal{E}_0$.

(b) $Q \cdot p \neq 0$.

(c) $\deg(Q) \leq r - D$, where $r = n/2 - n/(\log n)^c$ and D is as defined above.

Suppose we have such a Q then let $R = Q \cdot p$. Now R is a polynomial that vanishes on $\mathsf{No}^n_{n/(\log n)^c}$. This is because either p vanishes on $\mathsf{No}^n_{n/(\log n)^c} \setminus \mathcal{E}_0$ or Q vanishes on $\mathcal{E}_0$. Due to property (b), R is also a non-zero polynomial. Therefore using Fact 18, we know that it has degree at least r. Now assuming property (c) we get that p must have degree at least D, thereby proving the claim.

The existence of such a Q can be proved using arguments similar to those in [16]. The proof is omitted for lack of space. ◁

References

1 Leonard M. Adleman. Two Theorems on Random Polynomial Time. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 75–83, 1978.

2 Miklós Ajtai. $\sum^1{}_1$-formulae on finite structures. *Ann. Pure Appl. Logic*, 24(1):1–48, 1983. doi:10.1016/0168-0072(83)90038-6.

3 Miklós Ajtai and Michael Ben-Or. A Theorem on Probabilistic Constant Depth Computations. In *STOC*, pages 471–474. ACM, 1984.

4 Kazuyuki Amano. Bounds on the Size of Small Depth Circuits for Approximating Majority. In *ICALP (1)*, Lecture Notes in Computer Science, pages 59–70. Springer, 2009.

5 Sanjeev Arora and Boaz Barak. *Computational complexity : a modern approach*. Cambridge University Press, 2009.

6 Roger C. Baker, Glyn Harman, and János Pintz. The difference between consecutive primes, II. *Proceedings of the London Mathematical Society*, 83:532–562, 2001.

7 Joshua Brody and Elad Verbin. The Coin Problem and Pseudorandomness for Branching Programs. In *FOCS*, pages 30–39. IEEE Computer Society, 2010.

8 Gil Cohen, Anat Ganor, and Ran Raz. Two Sides of the Coin Problem. In *APPROX-RANDOM*, volume 28 of *LIPIcs*, pages 618–629. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

9 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.

10 Alexander Golovnev, Rahul Ilango, Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, and Avishay Tal. AC^0[p] Lower Bounds Against MCSP via the Coin Problem. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 66:1–66:15, 2019.

11 John Hastad. Almost Optimal Lower Bounds for Small Depth Circuits. *Advances in Computing Research*, 5:143–170, 1989.

12 Pavel Hrubes, Sivaramakrishnan Natarajan Ramamoorthy, Anup Rao, and Amir Yehudayoff. Lower Bounds on Balancing Sets and Depth-2 Threshold Circuits. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 72:1–72:14, 2019.

13 Svante Janson. Poisson Approximation for Large Deviations. *Random Struct. Algorithms*, 1(2):221–230, 1990.

14 Nutan Limaye, Karteek Sreenivasaiah, Srikanth Srinivasan, Utkarsh Tripathi, and S. Venkitesh. A fixed-depth size-hierarchy theorem for $\mathrm{AC}^0[\oplus]$ via the coin problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 442–453, 2019.

15 Ryan O'Donnell and Karl Wimmer. Approximation by DNF: Examples and Counterexamples. In *ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 195–206. Springer, 2007.

16 Igor Carboni Oliveira, Rahul Santhanam, and Srikanth Srinivasan. Parity Helps to Compute Majority. In *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA.*, pages 23:1–23:17, 2019.

17 Alexander A. Razborov. On the Method of Approximations. In *STOC*, pages 167–176. ACM, 1989.

18 Benjamin Rossman. On the constant-depth complexity of k-clique. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 721–730, 2008. doi:10.1145/1374376.1374480.

19 Benjamin Rossman and Srikanth Srinivasan. Separation of $\mathrm{AC}^0[\oplus]$ Formulas and Circuits. In *ICALP*, volume 80 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

20 Ronen Shaltiel and Emanuele Viola. Hardness Amplification Proofs Require Majority. *SIAM J. Comput.*, 39(7):3122–3154, 2010. doi:10.1137/080735096.

21 Roman Smolensky. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In *STOC*, pages 77–82. ACM, 1987.

22 Roman Smolensky. On Representations by Low-Degree Polynomials. In *FOCS*, pages 130–138. IEEE Computer Society, 1993.

23 Emanuele Viola. Randomness Buys Depth for Approximate Counting. *Computational Complexity*, 23(3):479–508, 2014. doi:10.1007/s00037-013-0076-6.

24 Andrew Chi-Chih Yao. Separating the Polynomial-Time Hierarchy by Oracles (Preliminary Version). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 1–10, 1985.

Planted Models for k-Way Edge and Vertex Expansion

Anand Louis
Indian Institute of Science, Bangalore, India
anandl@iisc.ac.in

Rakesh Venkat
Indian Institute of Technology, Hyderabad, India
rakeshvenkat@iith.ac.in

Abstract

Graph partitioning problems are a central topic of study in algorithms and complexity theory. Edge expansion and vertex expansion, two popular graph partitioning objectives, seek a 2-partition of the vertex set of the graph that minimizes the considered objective. However, for many natural applications, one might require a graph to be partitioned into k parts, for some $k \geqslant 2$. For a k-partition $S_1, \dots, S_k$ of the vertex set of a graph $G = (V, E)$, the *k-way edge expansion* (resp. vertex expansion) of $\{S_1, \dots, S_k\}$ is defined as $\max_{i \in [k]} \Phi(S_i)$, and the balanced k-way edge expansion (resp. vertex expansion) of G is defined as

$$\min_{\{S_1,\dots,S_k\} \in \mathcal{P}_k} \max_{i \in [k]} \Phi(S_i)\,,$$

where $\mathcal{P}_k$ is the set of all balanced k-partitions of V (i.e each part of a k-partition in $\mathcal{P}_k$ should have cardinality $|V|/k$), and $\Phi(S)$ denotes the edge expansion (resp. vertex expansion) of $S \subset V$. We study a natural planted model for graphs where the vertex set of a graph has a k-partition $S_1, \dots, S_k$ such that the graph induced on each S_i has large expansion, but each S_i has small edge expansion (resp. vertex expansion) in the graph. We give bi-criteria approximation algorithms for computing the balanced k-way edge expansion (resp. vertex expansion) of instances in this planted model.

2012 ACM Subject Classification Theory of computation → Semidefinite programming; Theory of computation → Graph algorithms analysis

Keywords and phrases Vertex Expansion, k-way partitioning, Semi-Random models, Planted Models, Approximation Algorithms, Beyond Worst Case Analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.23

Related Version A full version of the paper is available at `https://arxiv.org/abs/1910.08889`.

Funding *Anand Louis*: Supported in part by SERB Award ECR/2017/003296 and a Pratiksha Trust Young Investigator Award.

1 Introduction

The complexity of computing various graph expansion parameters are central open problems in theoretical computer science, and in spite of many decades of intensive research, they are yet to be fully understood [6, 5, 22, 7, 13, 40]. A central problem in the study of graph partitioning is that of computing the sparsest edge cut in a graph. For a graph $G = (V, E)$, we define the *edge expansion* of a set S of vertices, denoted by $\phi(S)$ as

$$\phi(S) \overset{\text{def}}{=} \frac{|E(S, V \setminus S)|}{|S|\,|V \setminus S|}\,|V|\,, \tag{1.1}$$

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 23; pp. 23:1–23:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where $E(S, V \setminus S) \overset{\text{def}}{=} \{\{u,v\} \in E | u \in S, v \in V \setminus S\}$. The edge expansion of the graph G is defined as $\phi_G \overset{\text{def}}{=} \min_{S \subset V} \phi(S)$. Related to this is the notion of the vertex expansion of a graph. For a graph $G = (V, E)$, we define the *vertex expansion* of a set S of vertices, denoted by $\phi^{\mathsf{V}}(S)$ as

$$\phi^{\mathsf{V}}(S) \overset{\text{def}}{=} \frac{|N(S) \cup N(V \setminus S)|}{|S|\,|V \setminus S|}\,|V| \,, \tag{1.2}$$

where $N(S) \overset{\text{def}}{=} \{v \in V \setminus S | \exists u \in S \text{ such that } \{u,v\} \in E\}$. The vertex expansion of the graph G is defined as $\phi^{\mathsf{V}}_G \overset{\text{def}}{=} \min_{S \subset V} \phi^{\mathsf{V}}(S)$. A few other related notions of vertex expansion have been studied in the literature, we discuss them in Section 1.4. We also give a brief description of related works in Section 1.4.

Graph k-partitioning

The vertex expansion and edge expansion objectives seek a 2-partition of the vertex set of the graph. However, for many natural applications, one might require a graph to be partitioned into k parts, for some $k \geqslant 2$. Let us use Φ to denote either ϕ (edge expansion) or ϕ^{V} (vertex expansion). For a k-partition $S_1, \ldots, S_k$ of the vertex set, the *k-way edge/vertex expansion* of $\{S_1, \ldots, S_k\}$ is defined as

$$\Phi^k\left(S_1, \ldots, S_k\right) \overset{\text{def}}{=} \max_{i \in [k]} \Phi(S_i) \,,$$

and the *k-way edge/vertex expansion* of G is defined as

$$\Phi^k_G \overset{\text{def}}{=} \min_{\{S_1, \ldots, S_k\} \in \mathcal{P}_k} \Phi^k\left(S_1, \ldots, S_k\right) \,,$$

where $\mathcal{P}_k$ is the set of all k-partitions of the vertex set. Optimizing these objective function is useful when one seeks a k-partition where each part has small expansion. The edge expansion version of this objective has been studied in [26, 23, 21], etc., and the vertex expansion version of this objective has been studied in [12]; see Section 1.4 for a brief summary of the related work.

For many NP-hard optimization problems, simple heuristics work very well in practice, for e.g. SAT [9], sparsest cut [18, 19], etc. One possible explanation for this phenomenon could be that instances arising in practice have some inherent structure that makes them "easy". Studying natural random/semi-random families of instances, and instances with planted solutions has been a fruitful approach towards understanding the structure of easy instances, and in modelling instances arising in practice, especially for graph partitioning problems [33, 29, 30, 28] (see Section 1.4 for a brief survey). Moreover, studying semi-random and planted instances of a problem can be used to better understand what aspects of a problem make it "hard". Therefore, in an effort to better understand the complexity of graph k-partitioning problems, we study the k-way edge and vertex expansion of a natural planted model of instances. We give bi-criteria approximation algorithms for instances from these models.

1.1 k-way planted models for expansion problems

We study the following model of instances.

▶ **Definition 1.1** (k-Part-edge). *An instance of k-Part-*edge*$(n, k, \varepsilon, \lambda, d, r)$ is generated as follows.*

1. *Let V be a set of n vertices. Partition V into k sets $\{S_1, S_2, \dots S_k\}$, with $|S_t| = n/k$ for every $t \in [k]$. For each $t \in [k]$, add edges between arbitrarily chosen pairs of vertices in S_t to form an arbitrary roughly d-regular (formally, the degree of each vertex should lie in $[d, rd]$) graph of spectral gap (defined as the second smallest eigenvalue of the normalized Laplacian matrix of the graph, see Section 2.1 for definition) at least λ.*
2. *For all $i, j \in [k]$, add edges between arbitrarily chosen pairs of vertices in $S_i \times S_j$ such that $\phi_G(S_i) \leqslant \varepsilon rd \; \forall i \in [k]$.*
3. *(Monotone Adversary) For each $t \in [k]$, add edges between any number of arbitrarily chosen pairs of vertices within S_t.*

Output the resulting graph G.

Analogously, we define the vertex expansion model.

▶ **Definition 1.2** (k-Part-vertex)**.** *An instance of k-Part-vertex$(n, k, \varepsilon, \lambda, d, r)$ is generated as follows.*

1. *Let V be a set of n vertices. Partition V into k sets $\{S_1, S_2, \dots S_k\}$, with $|S_t| = n/k$ for every $t \in [k]$. For each $t \in [k]$, add edges between arbitrarily chosen pairs of vertices in S_t to form an arbitrary roughly d-regular (formally, the degree of each vertex should lie in $[d, rd]$) graph of spectral gap (defined as the second smallest eigenvalue of the normalized Laplacian matrix of the graph, see Section 2.1 for definition) at least λ.*
2. *For each $t \in [k]$, partition S_t into T_t and $S_t \setminus T_t$ such that $|T_t| \leqslant \varepsilon n/k$. Add edges between any number of arbitrarily chosen pairs of vertices in $\cup_{i \in [k]} T_i$.*
3. *(Monotone Adversary) For each $t \in [k]$, add edges between any number of arbitrarily chosen pairs of vertices within S_t.*

Output the resulting graph G.

The only difference between k-Part-edge and k-Part-vertex is in the expansion of the sets. In step 2 of Definition 1.1, we ensured that $\phi(S_i) \leqslant \varepsilon rd \; \forall i \in [k]$[1]. In step 2 of Definition 1.2, the definition ensures that $\phi^{\mathsf{V}}(S_i) \leqslant \varepsilon k \; \forall i \in [k]$.

Both these models can be viewed as the generalization to k-partitioning of models studied in the literature for 2-partitioning problems for edge expansion [29], etc. and vertex expansion [28], etc. These kinds of models can be used to model communities in networks, where k is the number of communities. The intra-community connections are typically stronger than the inter-community connections. This can be modelled by requiring S_i to have large expansion (see Theorem 1.3 and Theorem 1.4 for how large a λ is needed compared to ε). Our work for $k > 2$ can be used to study more general models of communities than the case of $k = 2$.

1.2 Our Results

We give bi-criteria approximation algorithms for the instances generated from the k-Part-edge and k-Part-vertex models. We define OPT as follows

$$\mathsf{OPT} \stackrel{\text{def}}{=} \min_{\{P_1, \dots, P_k\} \in \widetilde{\mathcal{P}}_k} \Phi^k (P_1, \dots, P_k) \;,$$

where Φ is ϕ for k-Part-edge, and ϕ^{V} for k-Part-vertex, and $\widetilde{\mathcal{P}}_k$ is the set of all *balanced* k-partitions of the vertex-set, i.e. for each $\{P_1, \dots, P_k\} \in \widetilde{\mathcal{P}}_k$, we have $|P_i| = n/k \; \forall i \in [k]$. We note that in k-Part-edge, OPT $\leqslant \varepsilon rd$, and in k-Part-vertex, OPT $\leqslant \varepsilon k$.

[1] Since $\phi(S)$ measures the weight of edges leaving S (see (1.1)), it is often more useful to compare edge expansion to some quantity related to the degrees of the vertices inside S. Therefore, in step 2 of Definition 1.1, we require $\phi(S_i) \leqslant \varepsilon rd \; \forall i \in [k]$, instead of $\phi(S_i) \leqslant \varepsilon \; \forall i \in [k]$.

▶ **Theorem 1.3.** *There exist universal constants $c_1, c_2 \in \mathbb{R}^+$ satisfying the following: there exists a polynomial-time algorithm that takes as input a graph from the class k-Part-*$\mathsf{edge}(n, k, \varepsilon, \lambda, d, r)$ *with $\varepsilon \leqslant \lambda/(800kr^3)$, and outputs k disjoint sets of vertices $W_1, \ldots, W_k \subseteq V$, that for each $i \in [k]$ satisfy:*

1. $|W_i| \geqslant c_1 n/k$,
2. $\phi(W_i) \leqslant c_2 k\mathit{OPT}$.

▶ **Theorem 1.4.** *There exist universal constants $c_1, c_2 \in \mathbb{R}^+$ satisfying the following: there exists a polynomial-time algorithm that takes as input a graph from the class k-Part-*$\mathsf{vertex}(n, k, \varepsilon, \lambda, d, r)$ *with $\varepsilon \leqslant \lambda/(800kr^3)$, and outputs k disjoint sets of vertices $W_1, \ldots, W_k \subseteq V$, that for each $i \in [k]$ satisfy:*

1. $|W_i| \geqslant c_1 n/k$,
2. $\phi^{\mathsf{V}}(W_i) \leqslant c_2 k\mathit{OPT}$.

Note when $k = \mathcal{O}(1)$, Theorem 1.3 and Theorem 1.4 guarantee constant factor bi-criteria approximation algorithms. The currently best known approximation guarantees for general instances (i.e. worst case approximation guarantees) of k-way edge expansion problems are of the form $\mathcal{O}\left(\mathsf{OPT}\sqrt{\log n} f_1(k)\right)$ or $\mathcal{O}\left(\sqrt{\mathsf{OPT}} f_2(k)\right)$ where $f_1(k), f_2(k)$ are some functions of k, and the currently best known approximation guarantees for general instances (i.e. worst case approximation guarantees) of k-way vertex expansion problems are of the form $\mathcal{O}\left(\mathsf{OPT}\sqrt{\log n} f_3(k)\right)$ or $\mathcal{O}\left(\sqrt{\mathsf{OPT}} f_4(k, d)\right)$ where $f_3(k)$ is some functions of k and f_4 is some function of k and the maximum vertex degree d. We survey these results in Section 1.4. Note that our bi-criteria approximation guarantees in Theorem 1.3 and Theorem 1.4 are multiplicative approximation guarantees and are independent of n.

The above theorem shows that it is possible to produce k disjoint subsets, each of size $\Omega(n/k)$, each with expansion a factor k away from that of the planted partition. While this may not form a partition of the vertex set, it is not difficult to show that with a loss of a factor of k, we can indeed get a true partition. This idea of moving from disjoint sets to a partition is well-known, and has been used before in other works (for e.g., [21]).

▶ **Corollary 1.5.** *There exist universal constants $c_1, c_2 \in \mathbb{R}^+$ satisfying the following: there exists a polynomial-time algorithm that takes as input a graph from k-Part-*$\mathsf{edge}(n, k, \varepsilon, \lambda, d, r)$ *(resp. k-Part-*$\mathsf{vertex}(n, k, \varepsilon, \lambda, d, r)$*) with $\varepsilon \leqslant \lambda/800kcr^3$, and outputs a k-partition $\mathcal{P} = \{P_1, \ldots, P_k\}$ of V such that:*

1. *For each $i \in [k]$, $|P_i| \geqslant c_1 n/k$,*
2. *For each $i \in [k]$, $\phi(P_i) \leqslant c_2 k^2 \mathit{OPT}$ (resp. $\phi^{\mathsf{V}}(P_i) \leqslant c_2 k^2 \mathit{OPT}$).*

We note that the above result approximates the k-way expansion of the best *balanced* partition in G. The proofs of the above results are given in Section 3.

1.3 Proof Overview

For proving Theorem 1.3 and Theorem 1.4 we use an SDP relaxation (see Section 2.2) similar to the one used by [23, 31], etc. For the case when $k = 2$, [29, 28] used slightly different SDP constraints, and showed that when S_1 and S_2 contain large edge expanders, the set of SDP solution vectors $\{u_i : i \in V\}$ contain two sets L_1, L_2 such that $|L_1|, |L_2| = \Omega(n)$, L_1 and L_2 have small diameter, and the distance between L_1 and L_2 is $\Omega(1)$. The core of our analysis can be viewed as proving an analogue of this for $k > 2$ (Proposition 3.3), however, this requires some new ideas. For $i \in [k]$, let μ_i denote the mean of the vectors corresponding to the vertices in S_i. We use the expansion within S_i's together with the SDP constraints to show that for $i, j \in [k]$, $i \neq j$, each μ_i must have $\Omega(n/k)$ vertices sufficiently close to it, and that μ_i and μ_j must be sufficiently far apart. This can be used to show the existance of k such sets $L_1, \ldots, L_k$, such that for each $i \in [k]$, L_i has sufficiently small diameter and L_i is sufficiently far from L_j $\forall j \neq i$. The proof of our structure theorem is similar in spirit to the proof of structure theorem of [39], but our final guarantees are very different, we discuss their work in more detail in Section 1.4.

If we can compute k such sets $L_1, \ldots, L_k$, then using standard techniques, we can recover k sets having small expansion. In the case of $k = 2$, one could just guess a vertex from each these sets, and compute the two sets satisfying our requirements using standard techniques. For $k > 2$, guessing a vertex from each of the balls around μ_i would also suffice to compute sets $L_1, \ldots, L_k$ satisfying our requirements. However, doing this naively would take time $O(n^k)$. To obtain an algorithm for this task whose running time is $\mathcal{O}(\mathsf{poly}(n, k))$, we use a simple greedy algorithm (Algorithm 1) to iteratively compute the sets L_i such that L_i has sufficiently small diameter and is sufficiently far from L_j for all $j < i$. To ensure that this approach works, one has to ensure that at the start of iteration $i + 1$, the set of SDP vectors for the vertices in $V \setminus \cup_{j=i}^{i} L_i$ has at least $k - i$ clusters each of size $\Omega(n/k)$ and having small diameter. We use our structural result to prove that this invariant holds in all iterations of the algorithm.

1.4 Related Work

[28] studied the 2-way vertex-expansion in k-Part-vertex for $k = 2$, and gave a constant factor bi-criteria approximation algorithm. Our proofs and results can be viewed as generalizing their result to $k > 2$. They also studied a stronger semi-random model, and gave an algorithm for exact recovery (i.e. a 1-approximation algorithm) w.h.p. [29] studied the 2-way edge-expansion in a model similar to k-Part-edge for $k = 2$, and gave a constant factor bi-criteria approximation algorithm. Our proofs and results can be viewed as generalizing their result to $k > 2$.

k-partitioning problems. The minimum k-cut problem asks to find a k-partition of the vertex set which cuts the least number of edges; [43, 38, 42] all gave 2-approximation algorithms for this problem. A number of works have investigated k-way partitioning in the context of edge expansion. Bansal et al. [8] studied the problem of computing a k-partitioning $S_1, \ldots, S_k$ of the vertex set such that $|S_i| = n/k$ for each $i \in [k]$, which minimizes $\max_{i \in [k]} |E(S_i, V \setminus S_i)|$. They give an algorithm which outputs a k-partition of the vertex set $T_1, \ldots, T_k$ such that $|T_i| \leqslant (2 + \varepsilon)n/k$, and $\max_{i \in [k]} |E(T_i, V \setminus T_i)| \leqslant \mathcal{O}\left(\sqrt{\log n \log k}\right) \mathsf{OPT}$, where OPT denotes the cost of the optimal solution. There are also many connections between graph partitioning problems and graph eigenvalues. Let $0 = \lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_n$ denote the eigenvalues of the normalized Laplacian matrix of the graph. Typically, a different but related notion of edge expansion is used, which is defined as follows.

$$\phi'(S) \stackrel{\text{def}}{=} \frac{|E(S, V \setminus S)|}{\min\{\mathsf{vol}(S), \mathsf{vol}(V \setminus S)\}},$$

where $\mathsf{vol}(S)$ is defined as the sum of the degrees of the vertices in S. [25] gave an algorithm to find a k-partition which cuts at most $\mathcal{O}\left(\sqrt{\lambda_k} \log k\right)$ fraction of the edges. [21, 26] showed that for any k non-empty disjoint subsets $S_1, \ldots, S_k \subset V$, $\max_{i \in [k]} \phi'(S_i) = \Omega(\lambda_k)$. [21] (see also [26, 23]) gave an algorithm to find a $(1 - \varepsilon)k$ partition $S_1, \ldots, S_{(1-\varepsilon)k}$ of the vertex set satisfying $\max_i \phi'(S_i) = \mathcal{O}\left(\left(1/\varepsilon^3\right) \sqrt{\lambda_k \log k}\right)$ for any $\varepsilon > 0$, and a collection of k non-empty, disjoint subsets $S_1, \ldots, S_k \subset V$ satisfying $\max_i \phi'(S_i) = \mathcal{O}\left(k^2 \sqrt{\lambda_k}\right)$. [23] gave an algorithm to find a partition of V into $(1 - \varepsilon)k$ disjoint subsets $S_1, S_2, \ldots, S_{(1-\varepsilon)k}$, such that $\phi'(S_i) \leqslant \mathcal{O}\left(\sqrt{\log n \log k} \mathsf{OPT}\right)$.

Given a parameter δ, the small-set edge expansion problem asks to compute the set $S \subset V$ have the least edge expansion among all sets of cardinality at most $\delta \left|V\right|$ (or volume at most $\delta \mathsf{vol}\left(V\right)$). Bansal et al. [8] and Raghavendra et al. [41] gave a bi-criteria approximation algorithm for the small-set edge expansion problem. [23] gave an algorithm that outputs $(1-\varepsilon)k$ partition $S_1, \ldots, S_{(1-\varepsilon)k}$ such that $\max_i \phi'(S_i) = \mathcal{O}\left(\mathsf{poly}(1/\varepsilon)\sqrt{\log n \log k}\ \mathsf{OPT}\right)$, where OPT is least value of $\max_{i \in [k]} \phi'(S_i)$ over all k-partitions $S_1, \ldots, S_k$ of the vertex set. [23] also studied a balanced version of this problem, and gave bi-criteria approximation algorithms.

Let $\rho_k(G)$ denote $\min_{S_1,\ldots,S_k} \max_{i \in [k]} \phi'(S_i)$ where the minimum is over sets of k non-empty disjoint subsets $S_1, \ldots, S_k \subset V$. Kwok et al. [20] showed that for any $l > k$, $\rho_k(G) = \mathcal{O}\left(lk^6 \lambda_k / \sqrt{\lambda_l}\right)$. They also gave a polynomial time algorithm to compute non-empty disjoint sets $S_1, \ldots, S_k \subset V$ satisfying this bound. Combining this with the results of [21, 26], we get a $\mathcal{O}\left(lk^6/\sqrt{\lambda_l}\right)$ approximation to the problem of computing k non-empty disjoint subsets $S_1, \ldots, S_k \subset V$ which have the least value of $\max_{i \in [k]} \phi'(S_i)$. Here the approximation factor depends on λ_l, but even in the best case when $\lambda_l = \Omega(1)$ for some $l = O(k)$, the expression for the approximation guarantee reduces to $\mathcal{O}\left(k^7\right)$. They also show that for any $l > k$ and any $\varepsilon > 0$, there is a polynomial time algorithm to compute non-empty disjoint subsets $S_1, \ldots, S_{(1-\varepsilon)k} \subset V$ such that $\max_{i \in [(1-\varepsilon)k]} \phi'(S_i) = \mathcal{O}\left(\left(\left(l \log^2 k\right) / \left(\mathsf{poly}(\varepsilon)k\right)\right) \lambda_k / \sqrt{\lambda_l}\right)$.

Peng et al. [39] define the family of well clustered graphs to be those graphs for which $\lambda_{k+1}/\rho_k(G) = \Omega(k^2)$ (their structure theorem requires this ratio to be $\Omega(k^2)$, their algorithms require the separation to be larger, i.e. $\Omega(k^3)$) . They show that for such graphs, using the bottom k eigenvectors of the normalized Laplacian matrix, one can compute a k-partition which is close to the optimal k-partition for k-way edge expansion. They measure the closeness of their solution to the optimal solution in terms of the volume of the symmetric difference between the solution returned by their algorithm and the optimal solution. They start by showing that the vertex embedding of the graph into the k-dimensional space consisting of the bottom-k eigenvectors is clustered. Our technique to prove our main structural result Proposition 3.3, which shows that the SDP solution is clustered, is similar in spirit. Firstly, we note that the results of [39] apply to edge expansion problems and not vertex expansion problems. Moreover, due to the action of the monotone adversary, the λ_{k+1} of instances from k-Part-edge could be very small in which case the results of [39] wouldn't be applicable.

[12] showed that for a hypergraph $H = (V, E)$, there exist $(1-\varepsilon)k$ disjoint subsets $S_1, \ldots, S_{(1-\varepsilon)k}$ of the vertex set such that $\max_i \phi(S_i) = \mathcal{O}\left(k^2 \mathsf{poly}\log(k)/e^{1.5}\right)\sqrt{\gamma_k \log r}$, where r is the size of the largest hyperedge, $\phi(S)$ denotes the hypergraph expansion of a set of vertices S, γ_k is the kth smallest eigenvalue of the hypergraph Laplacian operator (we refer the reader to [12] for the definition of $\phi(\cdot)$, γ_k, etc.) Combining these ideas from [12] with the ideas from [24], we believe it should be possible to obtain an algorithm that outputs $(1-\varepsilon)k$ disjoint subsets $S_1, \ldots, S_{(1-\varepsilon)k}$ such that $\max_i \phi(S_i) = \mathcal{O}\left(k^2 \mathsf{poly}\log(k)\mathsf{poly}(1/\varepsilon)\right)\sqrt{\log n}\,\mathsf{OPT}$, where is OPT is least value of $\max_{i \in [k]} \phi(S_i)$ over all k-partitions $S_1, \ldots, S_k$ of the vertex set. Using a standard reduction from vertex expansion in graphs to hypergraph expansion, we get analogs of the above mentioned results for vertex expansion in graphs.

Vertex Expansion. An alternative, common definition of vertex expansion that has been studied in the literature is $\phi^{\mathsf{V,a}}(S) \overset{\text{def}}{=} \left(|V|\,|N(S)| \,/\, (|S|\,|V \setminus S|)\right)$, and as before, $\phi_G^{\mathsf{V,a}} \overset{\text{def}}{=} \min_{S \subset V} \phi^{\mathsf{V,a}}(S)$. As Louis et al. [27] show, the computation ϕ_G^{V} and $\phi_G^{\mathsf{V,a}}$ is equivalent upto constant factors.

Feige et al. [13] gave a $\mathcal{O}\left(\sqrt{\log n}\right)$-approximation algorithm for computing the vertex expansion of a graph. Bobkov et al. [10] gave a Cheeger-type inequality for vertex expansion in terms of a parameter λ_∞, which plays a role similar to λ_2 in edge-expansion. Building on this, Louis et al. [27] gave an SDP based algorithm to compute a set having vertex expansion at most $\mathcal{O}\left(\sqrt{\phi_G^{\mathsf{V}} \log d}\right)$ in graphs having vertex degrees at most d. This bound is tight upto constant factors [27] assuming the SSE hypothesis. Louis and Makarychev [24] gave a bi-criteria approximation for small-set vertex expansion.

Edge Expansion. Arora et al. [7] gave a $\mathcal{O}\left(\sqrt{\log n}\right)$-approximation algorithm for computing the edge expansion of a graph. Cheeger's inequlity [6, 5] says that $\lambda_2/2 \leqslant \min_{S \subset V} \phi'(S) \leqslant \sqrt{2\lambda_2}$.

Stochastic Block Models and Semi-Random Models. Stochastic Block Models (SBMs) are randomized instance-generation models based on the edge expansion objective and have been intensively studied in various works, starting with [16, 11, 17]. The goal is to identify and recover communities in a given random graph, where edges within communities appear with a probability p that is higher than the probability q of edges across communities. Both exact and approximate recovery guarantees for SBMs have been investigated using various algorithms [33, 35, 32, 1, 36, 37], leading to the resolution of a certain conjecture regarding for what range of model parameters are recovery guarantees are possible. While the above results deal mostly with the case of SBMs with two communities, k-way SBMs (for $k > 2$ communities) have been studied in recent works [2, 3, 4].

Semi-Random Models allow instance generation using a combination of both random edges and some amount of monotone adversarial action (i.e. not change the underlying planted solution). SDP-based methods seem to work well in this regard, since they are robust to such adversarial action. Many variants of semi-random models for edge expansion have been studied in literature. Examples include works due to Feige and Kilian [14], Guedon and Vershynin [15], Moitra et al. [34], and Makarychev et al. [29, 30, 31]. [31] also allows for a small amount of non-monotone errors in their model. These works give approximate and exact recovery guarantees for a range of parameters in their respective models.

2 Preliminaries and Notation

2.1 Notation

We denote graphs by $G = (V, E)$, where the vertex set V is identified with $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots n\}$. The vertices are indexed by i, j. For any $S \subseteq V$, we denote the induced subgraph on S by $G[S]$. Given $i \in V$ and $T \subseteq V$, define $N_T(i) \stackrel{\text{def}}{=} \{j \in T \ : \ \{i, j\} \in E\}$, and $N(i) = N_V(i)$.

Given the normalized Laplacian $\mathcal{L} = I - D^{-1/2}AD^{-1/2}$, the *spectral gap* of G denoted by λ, is the second-smallest eigenvalue of $\mathcal{L}$. *Spectral expanders* are a family of graphs with λ at least some constant (independent of the number of vertices in G).

Specific to graphs G generated in the k-Part-vertex and k-Part-edge models, let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be the collection of sets for any $i \in V$, let $S(i)$ denote the set $S \in \mathcal{S}$ such that $i \in S$. For a single subset $W \subseteq V$, we define $\partial W = \{i \in W : \exists j \notin W \text{ with } j \in N(i)\} \cup \{i \notin W : \exists j \in W \text{ with } j \in N(i)\}$, i.e., the *symmetric vertex* boundary of the cut $(W, V \setminus W)$. We let $E(\partial S)$ be the edges going across the cut $(S, V \setminus S)$, for any $S \subseteq V$. Given any k-partition of the vertex set $\mathcal{W} = \{W_1, \ldots, W_k\}$, we define $\partial \mathcal{W} = \cup_{i \in [k]} \partial W_i$ to be the set of boundary vertices on this partition, and $E(\partial \mathcal{W}) = \cup_{i \in [k]} E(\partial W_i)$ to be the edges across this partition.

2.2 SDP for k-way edge and vertex expansion

Our algorithms for both k-Part-edge and k-Part-vertex models use a natural semi-definite programming (SDP) relaxation for k-way expansion. The objective function we use is the "min-sum" objective in each case. For k-Part-vertex , it looks to minimize the number of boundary vertices in a balanced k-way partition of the vertex set, and correspondingly in k-Part-edge, the total number of edges across a balanced k-way partition of the vertex set.

For the k-Part-edge model, we use the following SDP relaxation.

▶ **SDP 2.1** (Primal). *k-Part-edge*

$$\min_U \quad \frac{1}{2}\sum_{i,j\in E} U_{ii} + U_{jj} - 2U_{ij}$$

subject to

$$\begin{aligned} U_{ii} &= 1 && \forall i \in V \\ U_{ij} &\geqslant 0 && \forall i,j \in V \\ \sum_j U_{ij} &= n/k && \forall i \in V \\ U_{jj} &\geqslant U_{ij} + U_{jk} - U_{ik} && \forall i,j,k \in V \\ U &\succeq 0 \end{aligned}$$

▶ **SDP 2.2** (Primal). *k-Part-vertex*

$$\min_U \quad \sum_{i\in V} \eta_i$$

subject to

$$\begin{aligned} \eta_i &\geqslant U_{ii} + U_{jj} - 2U_{ij} && \forall i, \forall j \in N(i) \\ U_{ii} &= 1 && \forall i \in V \\ U_{ij} &\geqslant 0 && \forall i,j \in V \\ \sum_j U_{ij} &= n/k && \forall i \in V \\ U_{jj} &\geqslant U_{ij} + U_{jk} - U_{ik} && \forall i,j,k \in V \\ U &\succeq 0 \end{aligned}$$

The intended integral solution for U in the SDP relaxation (SDP 2.2, SDP 2.1) for either model is $U_{ij} = 1$, if i, j lie in the same subset in the planted k-partition of V, and 0 otherwise. We can alternatively view the SDP variables as a set of vectors $\{u_i \in \mathbb{R}^n\}_{i\in V}$, satisfying $u_i^T u_j = U_{ij}$. These can be obtained by the Cholesky decomposition of the matrix U. Notice that the constraint $\sum_j U_{ij} = n/k$ in the relaxations above is specific to k-way partitions with exactly n/k vertices in each partition, and hence is satisfied by both models for the integral solution. The second-to-last set of constraints in either SDP are called ℓ_2^2 triangle inequalities, and can be rephrased in the language of vectors as:

$$\|u_i - u_j\|^2 + \|u_k - u_j\|^2 \geqslant \|u_i - u_k\|^2 \qquad \forall i,j,k \in V \tag{2.1}$$

It is easy to verify that these are satisfied by the ideal integral solution, corresponding to $u_i = e_t$, where $i \in S_t$.

For k-Part-edge, for every edge across the partition we accumulate a value of 1 in the SDP objective in the integral solution. Since every S_t has $\phi(S_t) \leqslant \varepsilon r d$, we have:

$$|E(\partial S_t)| \leqslant \varepsilon r d \frac{n}{k} \cdot (1 - \frac{1}{k}) \leqslant \varepsilon r d \frac{n}{k}$$
$$\implies 2\left|\cup_{t=1}^{k} E(\partial S_t)\right| \leqslant \varepsilon r d n$$

Since the number of edges going across the partition is at most[2] $\varepsilon r d n$, this is an upper bound on the optimum of SDP 2.1.

For k-Part-vertex , the integral solution will further set, $\eta_i = 2$ for any boundary vertex i of the partition $\mathcal{S}$, and $\eta_i = 0$ if i is not a boundary vertex, yielding a primal objective value of $2\varepsilon n$. Thus, the optimal value of SDP 2.2 is at most $2\varepsilon n$.

Furthermore, if OPT is as defined in Section 1.2, then in either case we have that SDP $\leqslant$ OPT $\cdot\, n$.

We introduce some notation regarding the SDP solution vectors $\{u_i\}_{i\in V}$ that will be useful for proofs. Let $d(i,j) \overset{\text{def}}{=} \|u_i - u_j\|^2$. Due to inequalities (2.1), $d(\cdot,\cdot)$ is a metric. Given a set $L \subseteq V$, define $d(i,L) \overset{\text{def}}{=} \min_{j\in L} d(i,j)$. The ℓ_2^2 diameter of L is $\mathrm{diam}(L) = \max_{i,j\in L} d(i,j)$. A ball of ℓ_2^2 radius a around a point $x \in \mathbb{R}^n$ is defined as $B(x,a) \overset{\text{def}}{=} \{j \in V : d(j,x) \leqslant a\}$.

Further proof-specific notations are defined as and when they are needed in the respective sections.

3 Bi-criteria Guarantees in the Planted Model

We now give a proof of Theorem 1.3, Theorem 1.4 and Corollary 1.5. The main idea is to show that the SDP solution is clustered around k disjoint balls, each of which have a significant overlap with a distinct S_i, for $i \in [k]$. We can then extract out k sets greedily using an ℓ_1 line embedding.

In what follows, it is convenient to view the variables in the primal SDP as being vectors $u_i \in \mathbb{R}^n$ for each $i \in V$ that satisfy $u_i^T u_j = U_{ij}$.

The missing proofs for the results in this section are given in the full version of the paper.

3.1 Preliminary Lemmas

▶ **Lemma 3.1.** *Let $\delta \leqslant 1/100$ and $\alpha \leqslant 1$ be real numbers. Let $\{u_i\}_{i\in V}$ be a feasible SDP solution vector set for SDP 2.1 or SDP 2.2. Suppose there exists a set $L \subseteq V$ that satisfies:*

(a) $|L| \geqslant \alpha n$

(b) $\mathrm{diam}(L) \leqslant \delta$.

We have:

(a) *(Edge) If $\{u_i\}_{i\in V}$ is an optimal solution to SDP 2.1 with objective value βn, then there exists an $i \in L$, and $a \in [\delta, 1/50]$ such that $W \overset{\text{def}}{=} B(i,a)$ satisfies $\phi(W) \leqslant \mathcal{O}(\beta/\alpha)$.*

(b) *(Vertex) If $\{u_i\}_{i\in V}$ is an optimal solution to SDP 2.2 with objective value βn, then there exists an $i \in L$, and $a \in [\delta, 1/50]$ such that $W \overset{\text{def}}{=} B(i,a)$ satisfies $\phi^{\mathsf{V}}(W) \leqslant \mathcal{O}(\beta/\alpha)$.*

[2] We use a slightly loose upper bound for convenience, to match up parameters in our proofs with the k-Part-vertex model.

Part (a) of the above lemma follows from standard arguments in edge-expansion literature. Part (b) is a slight modification of [28, Lemma 3.1] [3]. We defer both proofs to the full version of the paper.

We next show that if the SDP solution is clustered into k disjoint, well-separated balls of small diameter, then we can iteratively use Lemma 3.1 to find k disjoint sets, each with small vertex or edge expansion.

▶ **Lemma 3.2.** *Let $\delta \leqslant \frac{1}{100}$ and $k \in \mathbb{Z}$ be large enough. Suppose the optimal SDP solution vectors $\{u_i\}_{i\in V}$ to SDP 2.1 (resp. SDP 2.2) yield an objective value of βn and satisfy the following properties:*

(a) *There exist disjoint sets $L_1, L_2, \ldots, L_k \subseteq V$, with $\operatorname{diam}(L_t) \leqslant \delta$,*

(b) *For each $t \in [k]$, and for some constant γ, we have $|L_t| \geqslant \gamma n/k$,*

(c) *For every $t \neq t'$, $d(L_t, L_{t'}) \geqslant 1/10$.*

Then, we can in polynomial time, find k disjoint sets $W_1, \ldots, W_k \subseteq V$ such that for every $t \in [k]$, $|W_t| \geqslant \gamma n/k$, and $\phi(W_t) \leqslant \mathcal{O}(\beta k/\gamma)$ (resp. $\phi^{\mathsf{V}}(W_t) \leqslant \mathcal{O}(\beta k/\gamma)$).

3.2 Showing that the SDP solution is clustered

We next show that for any input instance from the class k-Part-edge or k-Part-vertex with appropriate parameters, every feasible set of SDP solution vectors are clustered. Using Lemma 3.2, we can then immediately conclude the proof of Theorem 1.3 and Theorem 1.4.

Our main technical result is the following proposition.

▶ **Proposition 3.3.** *Let $\{u_i\}_{i\in V}$ be the optimal solution SDP 2.1 (resp. SDP 2.2) for an instance G from k-Part-edge$(n, k, \varepsilon, \lambda, d, r)$ (resp. k-Part-vertex$(n, k, \varepsilon, \lambda, d, r)$) with $\varepsilon k r^3/\lambda \leqslant 1/800$. Then, there exist sets $L_1, \ldots, L_k \subseteq V$ such that:*

(a) $\operatorname{diam}(L_t) \leqslant 1/100$,

(b) $\forall t \in [k] : \ |L_t \cap S_t| \geqslant n/2k$,

(c) $\forall t \neq t' : \quad d(L_t, L_{t'}) \geqslant 1/10$.

Proof of Proposition 3.3. We begin with the following lemma; the proof is given in the full version of the paper.

▶ **Lemma 3.4.** *Let $\{u_i\}_{i\in V}$ be the optimal solution to the SDP for an instance G from k-Part-vertex or k-Part-edge. For each $t \in [k]$, let $\mu_t = \mathbb{E}_{i\in S_t}[u_i]$. The following holds:*

(a) $\forall t \in [k] : \quad \mathbb{E}_{j\in S_t}[\|\mu_t - u_j\|^2] \leqslant \frac{k\varepsilon r^3}{\lambda}$

(b) $\quad 1 \geqslant \|\mu_t\|^2 \geqslant 1 - k\varepsilon r^3/\lambda$

(c) $\forall t \neq t' \quad \mu_t^T \mu_{t'} \leqslant k\varepsilon r^3/\lambda$

We use this to prove Proposition 3.3. For each $t \in [k]$, define $L_t \overset{\text{def}}{=} B(\mu_t, 1/400)$. Clearly, $\operatorname{diam}(L_t) \leqslant 1/100$.

Since the parameters for either k-Part model are assumed to satisfy $\varepsilon k r^3/\lambda \leqslant 1/800$, we have that for every $t \in [k]$, item (a) from Lemma 3.4 implies that $\mathbb{E}_{j\in S_t}[\|\mu_t - u_j\|^2] \leqslant k\varepsilon r^3/\lambda \leqslant 1/800$. We can now use Markov's inequality:

[3] References to the results and proofs in [28] are with respect to the full version of that paper, available currently as an arXiv preprint.

$$\begin{aligned}
\Pr_{j\in S_t}\left[\|\mu_t-u_j\|^2>\frac{1}{400}\right] &= \frac{|S_t\setminus(L_t\cap S_t)|}{|S_t|} && \ldots \text{ since } L_t \overset{\text{def}}{=} B(\mu_t,1/400)\\
\implies \frac{|L_t\cap S_t|}{|S_t|} &= 1-\Pr_{j\in S_t}\left[\|\mu_t-u_j\|^2>\frac{1}{400}\right]\\
&\geqslant 1-\frac{\mathbb{E}_{j\in S_t}[\|\mu_t-u_j\|^2]}{1/400}=\frac{1}{2}\\
\implies |L_t\cap S_t| &\geqslant \frac{n}{2k}
\end{aligned}$$

To prove item (c) of the lemma, we first prove the following claim:

▷ Claim 3.5.

$$\forall t\neq t' \qquad \|\mu_t-\mu_{t'}\|^2\geqslant\frac{9}{10}$$

Proof.

$$\begin{aligned}
\|\mu_t-\mu_{t'}\|^2 &= \|\mu_t\|^2+\|\mu_{t'}\|^2-2\mu_t^T\mu_{t'}\\
&\geqslant 1-\frac{k\varepsilon r^3}{\lambda}+1-\frac{k\varepsilon r^3}{\lambda}-2\times\frac{k\varepsilon r^3}{\lambda} && \ldots \text{ using Lemma 3.4}\\
&\geqslant 1-\frac{4k\varepsilon r^3}{\lambda}\geqslant\frac{19}{20}>\frac{9}{10} && \ldots \text{ since } \frac{k\varepsilon r^3}{\lambda}\leqslant\frac{1}{800}\ .
\end{aligned}$$

◁

From the definition of the sets $\{L_t\}_{t\in[k]}$, we will use the (plain Euclidean) triangle inequality and the above claim. Let $t\neq t'$. We know that $d(L_t,L_{t'})=d(i,i')$ for some $i\in L_t$ and $i'\in L_{t'}$. Using this:

$$\begin{aligned}
d(L_t,L_{t'}) &= d(i,i')\\
&= \|u_i-u_{i'}\|^2\\
&\geqslant (\|\mu_t-\mu_{t'}\|-\|\mu_t-u_i\|-\|\mu_t-u_{i'}\|)^2\\
&\qquad \ldots \text{ by triangle inequality on the point sequence } \mu_t\to i\to i'\to\mu_{t'}\\
&\geqslant\left(\|\mu_t-\mu_t'\|-\frac{1}{20}-\frac{1}{20}\right)^2 && \ldots\text{since } d(\mu_{t'},i'),\ d(\mu_t,i)\leqslant\sqrt{\frac{1}{400}}=\frac{1}{20}\\
&\geqslant\left(\frac{9}{10}-\frac{1}{10}\right)^2>\frac{1}{10}\,.
\end{aligned}$$

◀

Using the above, we now infer the proof of Theorem 1.3 and Theorem 1.4.

Proof of Theorem 1.3 and Theorem 1.4. Consider the optimal SDP solution vectors $\{u_i\}_{i\in V}$ for an instance G from k-Part-edge$(n,k,\varepsilon,\lambda,d,r)$ (resp. k-Part-vertex$(n,k,\varepsilon,\lambda,d,r)$), with the parameters satisfying the given conditions, and having an objective value of βn. Note that $\beta\leqslant\mathsf{OPT}$, as the SDP is a relaxation. Using Proposition 3.3, we infer the existence of sets $L_1,\ldots,L_k$ satisfying the conditions given. The SDP solution thus satisfies all the conditions of Lemma 3.2, with $\delta=\frac{1}{100}$ and $\gamma=1/2$, and therefore, we can find in polynomial time, k disjoint subsets $W_1,\ldots,W_k$: $|W_t|\geqslant n/2k$, and $\phi(W_t)\leqslant\mathcal{O}(\beta k)$, for every $t\in[k]$ for k-Part-edge, or correspondingly $\phi^{\mathsf{V}}(W_t)\leqslant\mathcal{O}(\beta k)$ for k-Part-vertex. Algorithm 1 describes the steps in the algorithm explicitly. ◀

Algorithm 1 Algorithm for rounding SDP solutions for k-Part-vertex (k-Part-edge) instances.

```
Input: $G = (V, E)$ from $k$-Part$(n, k, \varepsilon, \lambda, r)$ and an optimal SDP solution $\{u_i\}_{i \in V}$ on $G$
Output: Disjoint sets $W_1, \dots, W_k \subseteq V$ with $|W_t| \geqslant n/2k$
$C \leftarrow \emptyset$
for $t \in 1, \dots k$ do
    $W_t \leftarrow \emptyset$
    for $i \in V$ do
        for $r \in [1/100, 1/50)$ do                    ▷ Can be done in a discrete fashion
            $\hat{W} \leftarrow B(i, r)$
            If $\left|\hat{W}\right| < n/2k$ or $\hat{W} \cap C \neq \emptyset$ continue
            (For $k$-Part-edge): If $W_t = \emptyset$ or $\phi(W_t) > \phi(\hat{W})$ then $W_t \leftarrow \hat{W}$
            (For $k$-Part-vertex): If $W_t = \emptyset$ or $\phi^{\mathsf{V}}(W_t) > \phi^{\mathsf{V}}(\hat{W})$ then $W_t \leftarrow \hat{W}$
        end for
    end for
    $C \leftarrow C \cup W_t$
end for
return $W_1, \dots, W_t$
```

Proof of Corollary 1.5. The proof for both parts uses a technique to move from disjoint sets to partitions used before, for instance in [21]. Since previous works use it for edge expansion already, we state the proof for k-Part-vertex first.

For k-Part-vertex: We start with the sets $W_1, \dots, W_k$ from Theorem 1.4. From the definition of ϕ^{V}, we have:

$$\begin{aligned} |\partial W_t| &= |N(W_t)| + |N(V \setminus W_t)| \\ &\leqslant \mathcal{O}(1) \cdot \mathsf{OPT} \cdot k \cdot \frac{|W_t|\,|V \setminus W_t|}{n} = \mathcal{O}(k \cdot \mathsf{OPT}\,|W_t|) \qquad \forall t \in [k] \end{aligned}$$

Define the partition $\mathcal{P} = \{P_1, \dots, P_k\}$ as follows: $P_i = W_i$ if $i \neq k$, and $P_k = V \setminus \biguplus_{i \in [k-1]} W_i$. Clearly, we have:

$$|\partial P_k| \leqslant \left| \bigcup_{t=1}^{k-1} \partial W_t \right| \leqslant \mathcal{O}(k \cdot \mathsf{OPT} \sum_{t=1}^{k-1} |W_t|) \leqslant \mathcal{O}(kn \cdot \mathsf{OPT})$$

Above, the last inequality follows since the W_t's are all disjoint. Since $|P_k| \geqslant \Omega(n/k)$, and $|V \setminus P_k| \geqslant \Omega(n)$, we infer that $\phi^{\mathsf{V},\mathsf{k}}(\mathcal{P}) \leqslant \phi^{\mathsf{V}}(P_k) \leqslant \mathcal{O}\left(k^2 \cdot \mathsf{OPT}\right)$.

For k-Part-edge: The proof is very similar to the preceding one for k-Part-vertex, except we work with edges. Again, from the definition of ϕ, we have, for the sets given by Theorem 1.3:

$$|E(\partial W_t)| \leqslant \mathcal{O}(1) \cdot \mathsf{OPT} \cdot k \cdot \frac{|W_t|\,|V \setminus W_t|}{n} = \mathcal{O}(k \cdot \mathsf{OPT}\,|W_t|)$$

As before, we define $\mathcal{P} = \{P_1, \dots, P_k\}$ as follows: $P_i = W_i$ if $i \neq k$, and $P_k = V \setminus \biguplus_{i \in [k-1]} W_i$. From the above bound on $|E(\partial W_t)|$, we get that:

$$|E(\partial P_k)| = \mathcal{O}(k \cdot \mathsf{OPT} \sum_{t=1}^{k-1} |W_t|) = \mathcal{O}(kn \cdot \mathsf{OPT}),$$

giving that $\phi^k(\mathcal{P}) = \mathcal{O}(k^2 \cdot \mathsf{OPT})$. ◀

References

1 Emmanuel Abbe, Afonso S Bandeira, and Georgina Hall. Exact recovery in the stochastic block model. *IEEE Transactions on Information Theory*, 62(1):471–487, 2016.

2 Emmanuel Abbe and Colin Sandon. Community detection in general stochastic block models: Fundamental limits and efficient algorithms for recovery. In *IEEE 56th Annual Symp. on Foundations of Computer Science (FOCS), 2015*, pages 670–688. IEEE, 2015.

3 Emmanuel Abbe and Colin Sandon. Recovering communities in the general stochastic block model without knowing the parameters. In *Advances in neural information processing systems*, pages 676–684, 2015.

4 Naman Agarwal, Afonso S Bandeira, Konstantinos Koiliaris, and Alexandra Kolla. Multisection in the stochastic block model using semidefinite programming. In *Compressed Sensing and its Applications*, pages 125–162. Springer, 2017.

5 Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

6 Noga Alon and Vitali D Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.

7 Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), 2009. (Preliminary version in *36th STOC*, 2004). doi:10.1145/1502793.1502794.

8 Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 17–26. IEEE, 2011.

9 Roberto Battiti and Marco Protasi. Approximate Algorithms and Heuristics for MAX-SAT. In *Handbook of Combinatorial Optimization: Volume1–3*, pages 77–148, Boston, MA, 1999. Springer US. doi:10.1007/978-1-4613-0303-9_2.

10 Sergey Bobkov, Christian Houdré, and Prasad Tetali. λ_∞, Vertex Isoperimetry and Concentration. *Combinatorica*, 20(2):153–172, 2000.

11 Ravi B. Boppana. Eigenvalues and Graph Bisection: An Average-case Analysis. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 280–285, Washington, DC, USA, 1987. IEEE Computer Society. doi:10.1109/SFCS.1987.22.

12 T.-H. Hubert Chan, Anand Louis, Zhihao Gavin Tang, and Chenzi Zhang. Spectral Properties of Hypergraph Laplacian and Approximation Algorithms. *J. ACM*, 65(3):15:1–15:48, 2018. doi:10.1145/3178123.

13 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM Journal on Computing*, 38(2):629–657, 2008. doi:10.1137/05064299X.

14 Uriel Feige and Joe Kilian. Heuristics for semirandom graph problems. *Journal of Computer and System Sciences*, 63(4):639–671, 2001.

15 Olivier Guédon and Roman Vershynin. Community detection in sparse networks via Grothendieck's inequality. *Probability Theory and Related Fields*, 165(3-4):1025–1049, 2016.

16 Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

17 Mark Jerrum and Gregory B Sorkin. The Metropolis algorithm for graph bisection. *Discrete Applied Mathematics*, 82(1):155–175, 1998.

18 George Karypis and Vipin Kumar. Analysis of Multilevel Graph Partitioning. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, New York, NY, USA, 1995. ACM. doi:10.1145/224170.224229.

19 George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998. doi:10.1137/S1064827595287997.

20 Tsz Chiu Kwok, Lap Chi Lau, Yin Tat Lee, Shayan Oveis Gharan, and Luca Trevisan. Improved Cheeger's Inequality: Analysis of Spectral Partitioning Algorithms Through Higher Order Spectral Gap. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 11–20, New York, NY, USA, 2013. ACM. doi:10.1145/2488608.2488611.

21 James R Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM (JACM)*, 61(6):37, 2014.

22 Tom Leighton and Satish Rao. Multicommodity Max-flow Min-cut Theorems and Their Use in Designing Approximation Algorithms. *J. ACM*, 46(6):787–832, November 1999. doi:10.1145/331524.331526.

23 Anand Louis and Konstantin Makarychev. Approximation algorithm for sparsest k-partitioning. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1244–1255. SIAM, 2014.

24 Anand Louis and Yury Makarychev. Approximation Algorithms for Hypergraph Small-Set Expansion and Small-Set Vertex Expansion. *Theory of Computing*, 12(1):1–25, 2016. doi:10.4086/toc.2016.v012a017.

25 Anand Louis, Prasad Raghavendra, Prasad Tetali, and Santosh Vempala. Algorithmic extensions of Cheeger's inequality to higher eigenvalues and partitions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 315–326. Springer, 2011.

26 Anand Louis, Prasad Raghavendra, Prasad Tetali, and Santosh Vempala. Many sparse cuts via higher eigenvalues. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1131–1140. ACM, 2012.

27 Anand Louis, Prasad Raghavendra, and Santosh Vempala. The Complexity of Approximating Vertex Expansion. In *Proc. of the 54th Annual Symp. on Foundations of Computer Science*, FOCS '13, pages 360–369, Washington, DC, USA, 2013. IEEE Computer Society. doi:10.1109/FOCS.2013.46.

28 Anand Louis and Rakesh Venkat. Semi-random Graphs with Planted Sparse Vertex Cuts: Algorithms for Exact and Approximate Recovery. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 101:1–101:15, 2018. Full Version at: arXiv:1805.09747.

29 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Approximation Algorithms for Semi-random Partitioning Problems. In *Proc. of the 44th Annual ACM Symp. on Theory of Computing*, STOC '12, pages 367–384. ACM, 2012. doi:10.1145/2213977.2214013.

30 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Constant Factor Approximation for Balanced Cut in the PIE Model. In *Proc. of the 46th Annual ACM Symp. on Theory of Computing*, STOC '14, pages 41–49, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591841.

31 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Learning Communities in the Presence of Errors. In *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1258–1291, Columbia University, New York, New York, USA, 23–26 June 2016. PMLR. URL: http://proceedings.mlr.press/v49/makarychev16.html.

32 Laurent Massoulié. Community Detection Thresholds and the Weak Ramanujan Property. In *Proc. of the 46th Annual ACM Symp. on Theory of Computing*, STOC '14, pages 694–703, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591857.

33 Frank D. McSherry. Spectral Partitioning of Random Graphs. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 529–537, Washington, DC, USA, 2001. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=874063.875554.

34 Ankur Moitra, William Perry, and Alexander S Wein. How robust are reconstruction thresholds for community detection? In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 828–841. ACM, 2016.

35 Elchanan Mossel, Joe Neeman, and Allan Sly. Belief propagation, robust reconstruction and optimal recovery of block models. In *Conference on Learning Theory*, pages 356–370, 2014.

36 Elchanan Mossel, Joe Neeman, and Allan Sly. Consistency Thresholds for the Planted Bisection Model. In *Proc. of the 47th Annual ACM Symp. on Theory of Computing*, STOC '15, pages 69–75, New York, NY, USA, 2015. ACM. doi:10.1145/2746539.2746603.

37 Elchanan Mossel, Joe Neeman, and Allan Sly. A proof of the block model threshold conjecture. *Combinatorica*, 2017. doi:10.1007/s00493-016-3238-8.

38 J Naor and Yuval Rabani. Tree Packing and Approximating fc-Cuts. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, volume 103, page 26. SIAM, 2001.

39 R. Peng, H. Sun, and L. Zanetti. Partitioning Well-Clustered Graphs: Spectral Clustering Works! *SIAM Journal on Computing*, 46(2):710–743, 2017. doi:10.1137/15M1047209.

40 Prasad Raghavendra and David Steurer. Graph Expansion and the Unique Games Conjecture. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 755–764, New York, NY, USA, 2010. ACM. doi:10.1145/1806689.1806792.

41 Prasad Raghavendra, David Steurer, and Prasad Tetali. Approximations for the Isoperimetric and Spectral Profile of Graphs and Related Parameters. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 631–640, New York, NY, USA, 2010. ACM. doi:10.1145/1806689.1806776.

42 R Ravi and Amitabh Sinha. Approximating k-cuts using network strength as a lagrangean relaxation. *European Journal of Operational Research*, 186(1):77–90, 2008.

43 Huzur Saran and Vijay V Vazirani. Finding k cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995.

Online Non-Preemptive Scheduling to Minimize Maximum Weighted Flow-Time on Related Machines

Giorgio Lucarelli
LCOMS, University of Lorraine, Metz, France
giorgio.lucarelli@univ-lorraine.fr

Benjamin Moseley
Tepper School of Business, Carnegie Mellon University, USA
moseleyb@andrew.cmu.edu

Nguyen Kim Thang
IBISC, Univ. Paris-Saclay, France
kimthang.nguyen@univ-evry.fr

Abhinav Srivastav
IBISC, Univ. Paris-Saclay, France
abhinavsriva@gmail.com

Denis Trystram
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, France
trystram@imag.fr

Abstract

We consider the problem of scheduling jobs to minimize the maximum weighted flow-time on a set of related machines. When jobs can be preempted this problem is well-understood; for example, there exists a constant competitive algorithm using speed augmentation. When jobs must be scheduled non-preemptively, only hardness results are known. In this paper, we present the first online guarantees for the non-preemptive variant. We present the first constant competitive algorithm for minimizing the maximum weighted flow-time on related machines by relaxing the problem and assuming that the online algorithm can reject a small fraction of the total weight of jobs. This is essentially the best result possible given the strong lower bounds on the non-preemptive problem without rejection.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Online Algorithms, Scheduling, Resource Augmentation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.24

Funding *Benjamin Moseley*: Google Research Award and NSF grants CCF-1830711, CCF-1824303, and CCF-1733873
Nguyen Kim Thang: OATA ANR-15-CE40-0015-01
Abhinav Srivastav: OATA ANR-15-CE40-0015-01

1 Introduction

We study the problem of online scheduling non-preemptive jobs to minimize the maximum (or ℓ_∞-norm of the) weighted flow-time on related machines. Here, we are given a set of n independent jobs that arrive over time. Each job j has a processing requirement p_j and a weight w_j. In the *related machines* environment, each machine i has speed s_i and the time required to process j is equal to p_j/s_i. The scheduling algorithm makes online decisions for assigning each job to one of the machines. If a job j arrives at time r_j and completes its processing at time C_j, then its flow-time F_j is defined as $(C_j - r_j)$. We focus on the objective

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 24; pp. 24:1–24:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of minimizing the maximum weighted flow-time, *i.e.*, $\max_j w_j F_j$. This metric is often used in systems where jobs are prioritized according to their weights and every job needs to be completed in a reasonable amount of time after its release. The problem of minimizing the maximum flow-time is a natural generalization of the load-balancing problem where jobs arrive over time. This problem is also closely related to deadline scheduling problems.

In this paper, we are interested in designing a non-preemptive schedule whose performance is bounded in the worst-case model. In *non-preemptive* setting, a job, once started processing, must be executed without interruption until its completion time. This is in contrast to *preemptive* setting where a job can be stopped and later continued from where it left off without penalty. Several strong theoretical lower bounds are known for simple instances [9, 4]. In order to overcome this lower bounds, Kalyanasundaram and Pruhs [12] and Phillips et al. [15] proposed the analysis of scheduling algorithms in terms of the speed and machine augmentations, respectively. Together these augmentations are commonly referred to as resource augmentation. In a resource augmentation analysis, the idea is to either give the scheduling algorithm faster processors or extra machines in comparison to the adversary. For preemptive problems, these models have been quite successful in establishing theoretical guarantees on algorithms that achieve good performance in practice [5, 11, 3, 10, 17].

Choudhury et al. [7, 8] proposed a different model of resource augmentation where the online algorithm is allowed to reject a small fraction of the total weight of the incoming jobs, while the adversary must complete all jobs. Theoretically, this model can lead to the discovery of good online algorithms, even in the face of strong lower bounds [7, 13]. Practically, the model is useful for systems where it is assumed that clients loose interest in their job if they wait too long to be completed. Choudhury et al. [7] considered the problem of load balancing as well as the problem of minimizing the maximum weighted flow-time in the restricted assignment setting. In this setting, we are given a set of machines and each job j can only be scheduled on a subset M_j of the machines while its execution takes p_j time units. Even with speed augmentation, these problems admit strong lower bounds in both preemptive and non-preemptive settings. However, online preemptive algorithms that achieve a $\mathcal{O}(1)$-competitive ratio and reject a small fraction of the total weight of jobs have been presented in [7].

Prior works have left open the online *non-preemptive* scheduling problem of minimizing the maximum weighted flow time. Even on a single machine, the problem is not understood and no positive result is known. Moreover, even with speed augmentation, simple examples lead to strong lower bounds. However, recent works on the rejection model [13] give the possibility of creating algorithms with strong guarantees for the non-preemptive setting. Thus, an intriguing open question is whether there exists a constant competitive algorithm for the maximum weighted flow-time objective in the non-preemptive setting assuming that a small fraction of total weight of jobs can be rejected. In this paper, we affirmatively answer this question for the case of related machines by proving the following theorem.

▶ **Theorem 1.** *For the non-preemptive scheduling problem of minimizing the maximum weighted flow-time on related machines, there exists a $\mathcal{O}(1/\epsilon^9)$-competitive algorithm that rejects at most $O(\epsilon)$-fraction of the total weight of jobs, where $\epsilon \in (0, 1)$.*

1.1 Problem definition and notation

We are given a set $\mathcal{M}$ of m machines and a set $\mathcal{J}$ of n jobs that arrive online. Each machine i processes a job at speed s_i. We index the machines such that $s_1 \geq s_2 \geq \ldots s_m$. Each job j is characterized by its release time r_j, its processing requirement p_j and its weight w_j.

The processing requirement and the weight of j are known at its release time r_j. If j is processed on machine i, then it requires p_j/s_i time units. The goal is to schedule the jobs *non-preemptively*. Given a schedule $\mathcal{S}$, the completion time of a job j is denoted by $C_j^{\mathcal{S}}$. The weighted flow-time of j is defined as $w_j F_j^{\mathcal{S}} = w_j(C_j^{\mathcal{S}} - r_j)$, which is the weighted amount of time during which j remains in the system. The objective is to minimize the maximum (ℓ_∞-norm of) weighted flow-time, *i.e.*, $\max_j w_j F_j^{\mathcal{S}}$. We omit the superscripts if the schedule $\mathcal{S}$ is clearly defined by the context.

Let F denote the value of the offline optimal solution. Let ϵ be an arbitrary constant such that $\epsilon \in (0,1)$. We assume that all weights w_j are of the form $(1/\epsilon)^k$, where k is an integer. This can be assumed by *rounding down* weights to the nearest power of $\frac{1}{\epsilon}$ which affects the competitive ratio by a factor of at most $\left(\frac{1}{\epsilon}\right)$. After rounding, we say that a job j is of *class* k if $w_j = \left(\frac{1}{\epsilon}\right)^k$. Let K denote the largest weight class of any job. A job j is *valid* on machine i, iff it takes at most F/w_j time units on i, that is $\frac{p_j}{s_i} \leq \frac{F}{w_j}$.

1.2 Organization

In Section 2, we present the works related to our problem. Then, in Section 3, we present an offline algorithm for the scheduling problem of minimizing the maximum weighted flow-time on related machines. This algorithm is inspired by Anand et al. [2] and uses a small look-ahead, allows preemptions and does not respect the release dates. Specifically, we assume that the value F of the optimal weighted flow time is known to the algorithm. Since release dates are not respected, the algorithm creates an infeasible schedule. Later, in Section 4, we discuss how to convert this offline algorithm to an online algorithm respecting the non-preemptive requirement. Note that the release dates will be respected due to the online nature. Finally, we explain how to remove the assumption about knowing the value F in Section 5.

2 Related Work

We discuss first related works for the unweighted case. For a single machine, First-In-First-Out is an optimal algorithm for minimizing the maximum flow-time. For identical machines, Bender et al. [14] and Ambulh and Mastrolilli [1] showed that the algorithm that schedules the incoming jobs on the least loaded machine, is $(3-2/m)$-competitive. On related machines, Bansal et al. [4] showed that there exists a 13.5-competitive algorithm. This has recently been improved to a 12.5-competitive algorithm by Im et al [16]. All the above algorithms are non-preemptive and their results hold against both the preemptive and non-preemptive adversary. For the more general setting of unrelated machines, Anand et al. [2] gave an $\mathcal{O}(1/\epsilon)$-competitive algorithm with $(1+\epsilon)$-speed augmentation and this result fundamentally uses preemption.

In the presence of weights, only results in the preemptive setting are known for the problem of minimizing the maximum weighted flow-time. Bender et al. [14] showed a lower bound of $\Omega(P^{1/3})$ on the competitive ratio on single machine where P is the ratio of the minimum to maximum job size. This was later improved to $\Omega(P^{0.4})$ in [6]. Both these lower bounds also hold if P is replaced with the ratio of the maximum to minimum weight. In speed augmentation model, Bansal and Pruhs [5] showed that the Highest Density First policy is $(1+\epsilon)$-speed $\mathcal{O}(1/\epsilon^2)$-competitive on a single machine. Chekuri and Moseley [6] presented a $(1+\epsilon)$-speed $\mathcal{O}(1/\epsilon)$-competitive algorithm for parallel machines, while Anand et al. [2] proposed a $(1+\epsilon)$-speed $\mathcal{O}(1/\epsilon^3)$-competitive algorithm for related machines. In the rejection model, Choudhury et al. [7] presented an $\mathcal{O}(1/\epsilon^4)$-competitive algorithm for the restricted assignment settings when an ϵ-factor of the weight of the jobs can be rejected by the online algorithm.

3 An Offline Look-ahead Algorithm with Preemptions

In this section we assume that the value F of the optimal solution is given, preemptions are allowed and the release dates of the jobs are not necessarily respected. Intuitively, we show the following. For ease of explanation assume that all jobs have unit weight. We consider all the jobs released during a long interval of size $O(F/\epsilon)$. Since the maximum weighted flow-time is F, all such jobs must be scheduled within an interval of size $O(F/\epsilon + F)$ by the optimal solution. We show that by rejecting an $O(\epsilon)$-fraction of the total weight of jobs, an online algorithm can schedule all the remaining jobs in the interval of size $O(F/\epsilon)$. The algorithm below builds on this idea when jobs have different weights. This is inspired by the work of Anand et al. [2] where speed augmentation is used to achieve a similar effect. Recall that there exists a strong lower bound in the speed augmentation model. In rejection model, one needs to ensure that the algorithm rejects at most $O(\epsilon)$-fraction of jobs both in terms of weights and volume.

We allow our algorithm to reject some jobs. For each weight class k and integer ℓ, let $I(k,\ell)$ denote the interval $\left[\frac{\ell F \epsilon^k}{\epsilon^3}, \frac{(\ell+1)F\epsilon^k}{\epsilon^3}\right)$. We say that a job j belongs to type (k,ℓ) if it is of class k and $r_j \in I(k,\ell)$. Observe that intervals $I(k,\ell)$ form a nested set of intervals. Note that at least $\left(\frac{1}{\epsilon^3}\right)$ jobs that belong to class k or more, can be scheduled during an interval $I(k,\ell)$. The online algorithm $\mathcal{A}$ is defined to have the following rejection and scheduling policies.

Rejection policy. The rejection policy of $\mathcal{A}$ is described in Algorithm 1. The algorithm uses a simple rejection policy where it ensures that for each interval $I(k,\ell)$ the algorithm rejects at least $\epsilon^2/2$-fraction of volume of jobs and $\mathcal{O}(\epsilon)$-fraction of weight of jobs.

Algorithm 1 $R_{\mathcal{A}}(\mathcal{I}, F, \epsilon)$.

```
for $k = K$ to 1 do
    for $\ell = 1, 2, \ldots$ do
        $J(k,\ell)$ := the set of jobs of type $(k,\ell)$
        $D := \lfloor \epsilon^2 |J(k,\ell)| + \epsilon \sum_{\substack{I(k'\ell') \subseteq I(k,\ell): \\ k'=k+1}} |J(k',\ell')| \rfloor + \sum_{\substack{I(k',\ell') \subseteq I(k,\ell): \\ k' \geq k+2}} |J(k',\ell')|$
        Reject longest-$D$ jobs from $J(k,\ell)$
```

Scheduling policy. The scheduling policy of $\mathcal{A}$ is described in Algorithm 2. The algorithm uses the following order: it picks jobs in the decreasing order of their class, and within each class it goes by increasing order of its intervals. When considering a job j, the algorithm schedules j during the interval $I(k,\ell)$ on the slowest *valid* machine with enough free space. Jobs may be scheduled preemptively. This completes the description of the algorithm $\mathcal{A}$.

Algorithm 2 $S_{\mathcal{A}}(\mathcal{I}, F, \epsilon)$.

```
for $k = K$ to 1 do
    for $\ell = 1, 2, \ldots$ do
        for each non-rejected job $j$ of type $(k,\ell)$ do
            $m_j$ := the slowest machine for which $j$ is valid
            for $i := m_j, \ldots, 1$ do
                If there is at least $p_j/s_i$ free slots (preemptive) on machine $i$ during $I(k,\ell)$ then
                schedule $j$ on $i$ during the first such free slots.
```

3.1 Analysis of the Offline Algorithm

In this section, we prove that Algorithm $S_{\mathcal{A}}$ will always find enough space to schedule the non-rejected set of jobs in $R_{\mathcal{A}}$.

▶ **Theorem 2.** *Algorithm $S_{\mathcal{A}}$ outputs a preemptive schedule for the set of non-rejected jobs which ensures that each job that belongs to type (k, ℓ) is scheduled during $I(k, \ell)$. Note that schedule may process jobs before their release date.*

We prove this by contradiction. Let j^* be the first non-rejected job that algorithm $\mathcal{A}$ cannot schedule on some machine i. Then we will show that the value of the offline optimal solution is strictly greater than F, which contradicts our assumption on the knowledge of the value of optimal offline solution, F.

Assume that j^* is of type (k^*, ℓ^*). We build a set S of job recursively. Initially S just contains j^*. We add j' of type (k', ℓ') to S if there is already a job j of type (k, l) in S satisfying the following conditions:

1. $k' \geq k$.
2. $\mathcal{A}$ processes j' on a machine i which is valid for j as well.
3. $\mathcal{A}$ processes j' during the $I(k', \ell')$ such that $I(k', \ell') \subseteq I(k, \ell)$

For a machine i and interval $I(k, \ell)$, define the *machine-interval* $I_i(k, \ell)$ as the time interval $I(k, \ell)$ on machine i. We construct a set $\mathcal{I}_M$ of machine-intervals as follows: For every job $j \in S$ of type (k, ℓ), we add the interval $I_i(k, \ell)$ to $\mathcal{I}_M$ for all machines i such that j is valid for i.

▶ **Definition 3.** *We say that an interval $I_i(k, \ell) \in \mathcal{I}_M$ is* maximal *if there is no other interval $I_i(k', \ell')$ which contains $I_i(k, \ell)$.*

Observe that every job in S except j^* gets processed in one of machine-intervals in $\mathcal{I}_M$. Let $\mathcal{I}_X$ denote the set of maximal intervals in $\mathcal{I}_M$. We show that the jobs in S satisfy the following property.

▶ **Lemma 4.** *For any maximal interval $I_i(k, \ell) \in \mathcal{I}_X$, Algorithm $S_{\mathcal{A}}$ processes a job on at least $(1 - \epsilon^3)$-fraction of the interval on machine i.*

Proof. We prove this property holds whenever we add a new maximal interval to $\mathcal{I}_X$. Suppose this property holds at some point in time, and we add a new job j' to S. Let $j, k, \ell, j', k', \ell'$ be as in the description of S. Since $k' \geq k$ and j is valid for i, the interval set $\mathcal{I}_X$ already contains the interval $I_{i'}(k, \ell)$ for all $i' \leq i$. Hence the intervals $I_{i'}(k', \ell')$ cannot be maximal for any $i' \leq i$. Suppose an interval $I_{i'}(k', \ell')$ is maximal, where $i' > i$, and j' is valid for i'. Our algorithm would have considered scheduling j' on i' before going to i. Hence the machine i' is at least $|I_{i'}(k', \ell')| - p_j/s_{i'}$ amount busy scheduling other jobs from S. The lemma follows since $p_j/s_{i'} \leq F/w_j \leq F\epsilon^k$. ◀

▶ **Corollary 5.** *There are at least $(\frac{1}{\epsilon^3} - 1)$ jobs of class k or more scheduled for every $I_i(k, \ell) \in \mathcal{I}_X$.*

Proof. Recall that the size of the interval $I_i(k, \ell)$ is $\frac{F\epsilon^k}{\epsilon^3}$ and the size of the longest job scheduled in the interval $I_i(k, \ell)$ is $\epsilon^k F$. Combining these facts with Lemma 4 shows that the corollary holds. ◀

Next we associate the set of rejected jobs to the maximal intervals. Recall that $|I(k, \ell)|$ denote the length of the interval $I(k, \ell)$. Intuitively, we show that for each maximal interval $I_i(k, \ell) \in \mathcal{I}_X$, we can associate at least $\mathcal{O}(\epsilon^2)|I_i(k, \ell)|$ volume of jobs that are rejected by the algorithm $R_{\mathcal{A}}$ such that these jobs are of type (k', ℓ') where $I(k', \ell') \subseteq I(k, \ell)$. To this end, let R denote the set of job rejected by $R_{\mathcal{A}}$. Let $R(k, \ell) = \{j \in R : j \text{ is of type } (k', \ell') \text{ and } I(k', \ell') \subseteq I(k, \ell)\}$.

▶ **Lemma 6.** *There exists a function $\phi : R \to \mathcal{I}_X$ such that for every $I_i(k, \ell) \in \mathcal{I}_X$, it holds that $vol(\phi^{-1}(I_i(k, \ell))) \geq \frac{\epsilon^2}{4}|I_i(k, \ell)|$ and $\phi^{-1}(I_i(k, \ell)) \subseteq R(k, \ell)$ where $vol(Q)$ denotes the total volume of jobs in the set Q.*

Proof. Fix a maximum interval $I = I_i(k, \ell)$. Let k_{max} denote the maximum weight class of the job scheduled in I. Recall the intervals $I_i(k', \ell') \subseteq I_i(k, \ell)$ are nested.

We first form an $1/\epsilon$-ary tree where a node $v(k', \ell')$ represents the set of jobs of type (k', ℓ') scheduled in the interval I on i. The node $v(k', \ell')$ is the the ancestor of the node $v(k'+1, \ell'')$ iff $I_i(k'+1, \ell'') \subseteq I_i(k', \ell')$. The height of this tree is $k_{max} - k$. Note that some of the leaves can be empty. Therefore, we trim the tree such that leaves are non-empty. For this, we find an empty leave and remove it from the tree. We repeat this procedure until no empty leaves are present. Note that an intermediate node of the tree can be empty. Next, we consider the following cases depending upon the number of jobs in the leaves:

Case 1: *There are at least $1/\epsilon^2$-jobs in each leaf.* The algorithm $R_{\mathcal{A}}$ rejects at least $\epsilon^2/2$ number of jobs at each non-empty node of the tree. Let j be such a job rejected by $R_{\mathcal{A}}$ for some node in the tree, then we define $\phi(j) = I$ (i.e., associate rejected job j to interval I). Recall that $R_{\mathcal{A}}$ rejects longest jobs among jobs of fixed class. Thus, the total volume of jobs associated with the interval I is at least $\epsilon^2/2$ and the lemma holds.

Case 2: *If the number of jobs in each leaf is between $1/\epsilon$ and $1/\epsilon^2$.* The algorithm $R_{\mathcal{A}}$ rejects at least $\epsilon^2/2$ fraction of volume of jobs at each non-empty node except leaves. As before, let j be such a job rejected by $R_{\mathcal{A}}$, then we define $\phi(j) = I$. We show that the total volume of jobs in the leaves are small. Let $v(k', \ell')$ denote jobs corresponding to some leaf. Then $|v(k', \ell')| < 1/\epsilon^2$. The total volume of jobs in $v(k', \ell')$ is at most $(F\epsilon^{k'})/\epsilon^2 = \epsilon|I_i(k', \ell')|$. Observe that the jobs of any two leaves are scheduled independent of each other in separate sub-intervals. Combining this fact with the previous bound on the volume of jobs in leaves implies that the total volume of jobs in leaves is at most $\epsilon|I|$. Thus, the total volume of jobs scheduled during the interval I is at most $2.vol(\phi^{-1}(I))/\epsilon^2 + \epsilon|I|$. Since $S_{\mathcal{A}}$ processes jobs on at least $(1-\epsilon^3)$-fraction of I, it holds that $vol(\phi^{-1}(I)) \geq (\epsilon^2/2)(1 - \epsilon^3 - \epsilon)|I| \geq (\epsilon^2/4)|I|$.

Case 3: *If the number of jobs in each leaf is strictly less than $1/\epsilon$.* If the algorithm rejects ϵ^2-fraction of total volume of jobs at each non-empty level other than the leaf, then the lemma holds (the proof is similar to Case 2). Thus, we consider the case where the parent of a leaf does not reject ϵ^2-fraction of the total volume of jobs. This implies that each parent has at most $1/\epsilon^2$ number of jobs and the height of the subtree rooted at the parent node is at most 1. The algorithm $R_{\mathcal{A}}$ rejects $\epsilon^2/2$ jobs for each node from the root to the parent of parent of a leaf. As before, let j be such a job rejected by $R_{\mathcal{A}}$, then we define $\phi(j) = I$. Unlike Case 2 where intervals corresponding to leaves are disjoint, th intervals of parents of two leaves can overlap. Here, we use the top-down approach to count the total volume of jobs. Each job in the parent node is split into $1/\epsilon$-parts. We "virtually force" these parts to be accounted in the leaves of that parent (even though their weight class is strictly smaller than the weight class of the leaves). Thus the number of jobs in each leaf can increase by at most $1/\epsilon^2$. Using arguments similar to Case 2, the total volume of jobs in the leaves is at most $2\epsilon I$. Since $S_{\mathcal{A}}$ process job on at least $(1-\epsilon^3)$-fraction of I, it holds that $vol(\phi^{-1}(I))$ is at least $\epsilon^2/2(1 - \epsilon^3 - 2\epsilon)|I|$-volume of jobs. ◀

▶ **Corollary 7.** *The total volume of jobs in* $S' = S \cup R$ *is greater than* $\sum_{I(k,\ell) \in \mathcal{I}_{\mathcal{X}}} I(1+\epsilon^3)|I(k,\ell)|$.

▶ **Lemma 8.** *If the value of offline solution is at most* F*, then the total volume of jobs in* S' *is at most* $\sum_{I(k,\ell) \in \mathcal{I}_{\mathcal{X}}} (1+\epsilon^3)|I(k,\ell)|$.

Proof. For any maximal interval $I(k,\ell)$ on machine i, let $I_i^{\epsilon}(k,\ell)$ be the interval of length $(1+\epsilon^3)|I(k,\ell)|$ which starts at the same time as $I(k,\ell)$ on machine i.

Let $j \in S$ be a job of type (k,ℓ). The optimal offline solution must schedule j within $F\epsilon^k$ of its release date. Since $r_j \in I(k',\ell') \subseteq I(k,\ell)$, the optimal solution must process a job j during $I^{\epsilon}(k,\ell)$. So, the total volume of jobs in S can be at most $|\bigcup_{I(k,\ell) \in \mathcal{I}_{\mathcal{X}}} I_i^{\epsilon}(k,\ell)| \leq \sum_{I(k,\ell) \in \mathcal{I}_{\mathcal{X}}} (1+\epsilon^3)|I(k,\ell)|$. ◀

Clearly, Corollary 7 contradicts Lemma 8. So, Algorithm $S_{\mathcal{A}}$ must be able to process all the jobs.

▶ **Lemma 9.** *The total weight of jobs rejected by the algorithm* $R_{\mathcal{A}}$ *is* $\mathcal{O}(\epsilon)$*-fraction of the total weight of jobs in the instance* $\mathcal{I}$.

4 The Online Algorithm $\mathcal{B}$

The previous algorithm $\mathcal{A}$ is an offline preemptive algorithm for $\mathcal{I}$ that does not respect the release dates. This section presents an online non-preemptive algorithm $\mathcal{B}$. This algorithm is assumed to know the optimal objective F and this algorithm is extended in a later section to when this is not known. The algorithm maintains a queue for each machine i and time t. Unlike the previous algorithm, $\mathcal{B}$ rejects the job of type (k,ℓ) at the end of the interval $I(k,\ell)$. For each non-rejected job j, $\mathcal{B}$ uses $S_{\mathcal{A}}$ to figure out the assignment of jobs to the machines. This algorithm differs from the online algorithm mentioned in Anand et al. [2] as it schedules jobs non-preemptively and does not necessarily process jobs in their decreasing order of their weights.

The rejection and assignment policies of $\mathcal{B}$ in given Algorithm 3.

Algorithm 3 $M_{\mathcal{B}}(\mathcal{I}, F, \epsilon)$.

```
for $t = 0, 1, 2, \cdots$ do
  Let $K$ denote the largest class of a job.
  for $k = K$ to 1 do
    if $t$ is the end point of the interval $I(k,\ell)$ for some $\ell$ then

      Rejection similar to $R_{\mathcal{A}}$
      $J(k,\ell) :=$ the set of jobs of type $(k,\ell)$
      $D := \lfloor \epsilon^2|J(k,\ell)| + \epsilon \sum_{\substack{I(k'\ell') \subseteq I(k,\ell): \\ k'=k+1}} |J(k',\ell')| \rfloor + \sum_{\substack{I(k',\ell') \subseteq I(k,\ell): \\ k' \geq k+2}} |J(k',\ell')|$
      Reject longest-$D$ Tjobs from $J(k,\ell)$ from the remaining jobs in $J(k,\ell)$

    Assignment similar to $S_{\mathcal{A}}$
    for each non-rejected job $j$ of class $k$ do
      Let $m_j$ denote the machine on which $j$ is scheduled by $S_{\mathcal{A}}$
      Assign $j$ to the queue of $m_j$
```

After the execution of Algorithm 3, the algorithm $\mathcal{B}$ uses two more rejection policies for each machine i . The first policy ensures that $\mathcal{B}$ rejects $\mathcal{O}(\epsilon^2)$-fraction of new assigned jobs whereas the second policy ensures that $\mathcal{B}$ processes jobs in a non-preemptive fashion. At any time if the machine i is idle, $\mathcal{B}$ picks a job from the highest class according to the ordering given by $S_{\mathcal{A}}$.

Making $\mathcal{B}$ Non-preemptive. We now detail the second rejection policy. During the processing of a job of some class on a machine i, the algorithm maintains a bound on total weight of higher class jobs that are newly assigned to machine i. Let j be the job running at the start of interval $I(k, \ell+1)$ on machine i. Let k_j denote the class of j. $\mathcal{B}$ rejects j if there is a new job j' of type (k', l') that $k' \geq k_j + 2$ and the intervals $I(k', \ell')$ and $I(k, \ell)$ end at same time. This ensures that the weight of job j and j' differ at least by a factor of $1/\epsilon$. It may happen that there is no job class $k' \geq k_j + 2$. In this case, the algorithm $\mathcal{B}$ rejects j if there are at least $(1/\epsilon$ newly arrived jobs of type (k', ℓ') if $k' \geq k_j + 1$ and the intervals $I(k', \ell')$ and $I(k, \ell)$ end at same time. Note that this also ensures the weight of newly arrived jobs is at least an $(1/\epsilon)$ times the weight of current running job. These rejection policies and scheduling policy of the algorithm $\mathcal{B}$ for the machine i at time t is mentioned in Algorithm 4.

Algorithm 4 $S_{\mathcal{B}}(\mathcal{I}, F, \epsilon, i, t)$.

```
Rejection similar to $R_{\mathcal{A}}$
for $k = K$ to 1 do
    if $t$ is the end point of the interval $I(k, \ell)$ for some $\ell$ then
        $J_i(k, \ell) :=$ the set of jobs of type $(k, \ell)$ assigned to $i$ at $t$
        $D := \lfloor 2\epsilon^2 |J_i(k, \ell)| + 2\epsilon \sum_{\substack{I(k'\ell') \subseteq I(k, \ell): \\ k' = k+1}} |J_i(k', \ell')| \rfloor + 2 \sum_{\substack{I(k', \ell') \subseteq I(k, \ell): \\ k' \geq k+2}} |J_i(k', \ell')|$
        Reject $D$-longest jobs from $J(k, \ell, i)$
Making algorithm non-preemptive
Let $j \in (k, \ell)$ be the job executing on $i$ at $t$
Let $J_{k'}$ denote the set of jobs of class $k'$ assigned to $i$ at $t$
if $|J_{(k+1)}| \geq 1/\epsilon$ or $\exists k'' : |J_{(k'')}| > 0$ and $k'' \geq k + 2$ then
    Reject $j$
Scheduling Policy
if the machine $i$ is idle then
    Start processing the earliest job of highest class in the queue of $i$
```

4.1 Analysis

For a class k, let J_k be the jobs of class at least k. For a class k, integer ℓ, and machine i, let $Q(i, k, \ell)$ denote the jobs of J_k which are in the queue of machine i at the beginning of $I(k, \ell)$. The jobs in $Q(i, k, \ell)$ could consist of either

1. jobs in $Q(i, k, \ell - 1)$, or
2. jobs of J_k which get processed by $\mathcal{A}$ during $I(k, \ell - 1)$ on machine i. Indeed, the jobs of J_k which are dispatched to machine i during $I(k, \ell - 1)$ will complete processing in $I(k, \ell - 1)$ in $\mathcal{A}$ and hence may get added (if not rejected) to $Q(i, k, \ell)$. Let $P(i, k, \ell - 1)$ denotes the volume of such jobs that are added by $\mathcal{B}$ to the queue of machine i.

Next, we note some properties of the algorithm $\mathcal{B}$:

▶ **Property 1.** *A job j gets scheduled in $\mathcal{B}$ only in later slots than those it was scheduled on by $\mathcal{A}$.*

▶ **Property 2.** *For a class k, integer ℓ and machine i, the total processing of jobs in $P(i,k,\ell)$ is at most $\frac{(1-\epsilon^3)F\epsilon^k}{\epsilon^3}$.*

Proof. If the volume of jobs processed by algorithm $\mathcal{A}$ during the interval $I(k,\ell)$ is at most $\frac{(1-\epsilon^3)F\epsilon^k}{\epsilon^3}$, then the property holds trivially. Assume that the volume of jobs processed by algorithm $\mathcal{A}$ during the interval $I(k,\ell)$ is strictly greater than $\frac{(1-\epsilon^3)F\epsilon^k}{\epsilon^3}$. Then it holds that the algorithm rejects at least $\epsilon^2/4$-fraction of volume of jobs assigned to i (the proof is similar to Lemma 6). Thus the total volume of jobs assigned to i is strictly greater than $\frac{(1+\epsilon^3)F\epsilon^k}{\epsilon^3}$. But, this contradicts Theorem 2. ◀

▶ **Property 3.** *For a class k, integer l and machine i, the total remaining processing time of jobs in $Q(i,k,\ell)$ is at most $\frac{(1-\epsilon^3)F\epsilon^k}{\epsilon^3}$.*

Proof. We use induction. Suppose this is true for some i,k,l. We show that this holds for $i,k,\ell+1$ as well. By induction $|Q(i,k,\ell)|$ is at most $\frac{(1-\epsilon^3)F\epsilon^k}{\epsilon^3}$. We consider multiple separate cases based on which job gets processed during the interval $I(k,\ell)$ on machine i.

1. *Suppose the machine i is busy processing jobs from J_k during $I(k,\ell)$.*
   Then algorithm either processes job from $Q(i,k,\ell)$ or $P(i,k,\ell)$. The total volume of such jobs are bounded by $\frac{2(1-\epsilon^3)F\epsilon^k}{\epsilon^3}$. The property holds since the total volume of job processed by i is $|I(k,\ell)| = \frac{F\epsilon^k}{\epsilon^3}$.
2. *Suppose job j of class smaller than k is processed at the start of $I(k,\ell)$ and $Q(i,k,\ell)=0$.*
   In this case, $Q(i,k,\ell+1)$ consists of the jobs of $P(i,k,\ell)$. The property follows since $|P(i,k,\ell)| \leq \frac{(1-\epsilon^3)F\epsilon^k}{\epsilon^3}$.
3. *Suppose job j of class smaller than k is processing at the start of $I(k,\ell)$ and $Q(i,k,\ell)>0$.*
   We show that $Q(i,k,\ell)$ is at most $\frac{F\epsilon^k}{\epsilon}$. Let σ_j denote the starting time of job j on machine i. Then we have that $\sigma_j > \frac{\ell F\epsilon^k}{\epsilon^3} - p_j/s_i \geq \frac{\ell F\epsilon^k}{\epsilon^3} - \frac{F\epsilon^k}{\epsilon}$. Since algorithm $\mathcal{B}$ prefers jobs of higher class, it must be the case that at σ_j no job of class k or higher was available with machine i . Hence $Q(i,k,\ell)$ consists of jobs that were added to the queue of machine i during the interval $\left(\sigma_j, \frac{\ell F\epsilon^k}{\epsilon^3}\right]$. Since $Q(i,k,\ell)>0$ and the class of j is strictly smaller than k, j must belong of class $(k-1)$, otherwise $\mathcal{B}$ would reject j due to non-preemptive rejection policy. Moreover, there are at most $1/\epsilon$ jobs of class k in $Q(i,k,\ell)$. Hence, the total volume of jobs in $Q(i,k,\ell)$ is most $\frac{F\epsilon^k}{\epsilon}$. The property follows from the facts that $\mathcal{B}$ spends at most $\frac{F\epsilon^k}{\epsilon}$ processing time on j.
4. Suppose $\mathcal{B}$ processes a job of class smaller than k at some point in $I(k,\ell)$.
   This implies that $Q(i,k,\ell+1)$ contains jobs that are released during the interval $I(k,\ell)$. The property holds due to Claim 2. ◀

▶ **Theorem 10.** *In the schedule $\mathcal{B}$ a job j of class k has flow-time at most $\frac{F\epsilon^k}{\epsilon^8}$. Hence the algorithm $\mathcal{B}$ is an $O(\frac{1}{\epsilon^9})$-competitive algorithm that rejects at most $O(\epsilon)$-fraction of total weights of job.*

Proof. Consider a job j of class $(k,\ell-1)$. Suppose it gets processed on machine i. The algorithm $\mathcal{B}$ adds j to the queue $Q(i,k,\ell)$. Let j' be the job running at beginning of the interval $I(k,\ell)$. Property 3 from above implies that the total remaining processing time of jobs in $Q(i,k,\ell)$ is at most $\frac{(1-\epsilon^3)F\epsilon^k}{\epsilon^3} = (1-\epsilon^3)|(k,\ell)|$.

Consider an interval I that starts at same time as $I(k,\ell)$ and has length $\frac{(1-\epsilon^3)F\epsilon^k}{\epsilon^7} = |I(k,\ell)|/\epsilon^4$. During I, the algorithm process jobs of J_k that are either in (1) $Q(i,k,\ell)$, or (2) processed by $\mathcal{A}$ on machine i. From Property 2, the total processing of jobs in (2) is $(1-\epsilon^3)|I|$. This leaves us with $\epsilon^3|I|$ processing time. This is enough of process the jobs in $Q(i,k,\ell)$ and j' as $(1-\epsilon^3)\frac{F\epsilon^k}{\epsilon^3} + F\epsilon^{k-1} \leq \frac{F\epsilon^k}{\epsilon^4} = \epsilon^3|I|$. So the flow time of j is at most $|I| + |I(k,\ell)| = F\epsilon^k(\frac{1}{\epsilon^7} + \frac{1}{\epsilon^3})$. ◀

5 Removing the assumption about knowledge of F

In this section, we show how to remove the assumption about knowledge of F. We apply the standard double trick that is often used in the online algorithms. Recall that our previous look-ahead algorithm assumed that we know the optimal F. Here, we will construct another look-ahead algorithm $\mathcal{C}$ which will invoke $\mathcal{A}$ for different guesses of F. Fix an instance $\mathcal{I}$. Let $I(k,\ell,F)$ be the interval $[\frac{\ell F\epsilon^k}{\epsilon^3}, \frac{(\ell+1)F\epsilon^k}{\epsilon^3})$. This is same as $I(k,\ell)$ except that the intervals are also parameterized by F. Similarly, we say that a job of class k is of type (k,ℓ,F) if $r_j \in I(k,\ell,F)$.

Our algorithm will work with the guesses of F which are powers of $\left(\frac{1+\epsilon^3}{\epsilon^3}\right)$. Without the loss of generality, we assume that all release dates and processing times are integers. We first generalize the algorithm $\mathcal{A}$. The new algorithm, denoted by $\mathcal{A}'$, will take as parameters an instance I', the guess F and a starting time t_0 - all release dates in I' will be at least t_0. It will run $\mathcal{A}'$ with the understanding that time start at t_0. The interval $I(k,\ell,F)$ will be defined as $[t_0 + \frac{\ell F\epsilon^k}{\epsilon^3}, t_0 + \frac{(\ell+1)F\epsilon^k}{\epsilon^3})$. The algorithm $\mathcal{C}$ is described below.

Algorithm 5 A look-ahead algorithm $\mathcal{C}$.

```
Initialize $F_0 = 1$, $t_0 = 0$, $\mathcal{I}_0 = \mathcal{I}$
for $u = 0, 1, 2, \ldots$ do
    Run $\mathcal{A}'(\mathcal{I}_u, F_u, t_u)$
    if All non-rejected jobs are finished then
        Stop and output the scheduled produced.
    else
        let $j$ be the first non-rejected job which algorithm $\mathcal{A}'(\mathcal{I}_u, F_u, t_u)$ is not to schedule.
        Suppose $j$ is of type $(k, \ell, F_u)$.
        Define $t_{u+1}$ be the end-point of $I(k, \ell, F_u)$.
        Define $\mathcal{I}_{u+1}$ be the jobs in $\mathcal{I}_u$ which are not scheduled yet.
        Define $r_j = \max\{t_{u+1}, r_j\}, \forall j \in \mathcal{I}_{u+1}$.
        Set $F_{u+1} = F_u \left(\frac{1+\epsilon^3}{\epsilon^3}\right)$.
```

Note that this algorithm, like Algorithm $\mathcal{A}$, is preemptive.

5.1 Analysis

Suppose during some iteration u, we find a job j^* that the algorithm is not able to schedule in iteration u. Let j^* be type of (k^*, ℓ^*, F_u). Recall that t_{u+1} is the end point of $I(k^*, \ell^*, F_u)$. For a job $j \in \mathcal{I}_u$ let r_j^u denote its release date in $\mathcal{I}_u$.

▶ **Lemma 11.** *Any job $j \in \mathcal{I}_{u+1}$ with $r_j^u < t_{u+1}$ must be of class at most k^*. Further, if such a job is of class k, then $t_{u+1} - r_j \leq F_{u+1}\epsilon^k$.*

Proof. Suppose $j \in \mathcal{I}_u$ and $r_j^u < t_{u+1}$. If j is of type (k, ℓ, F_u) such that $k > k^*$, then $I(k, \ell, F_u) \subseteq I(k^*, \ell^*, F_u)$. Hence the interval $I(k, \ell, F_u)$ end at or or before t_{u+1}. By definition of j^* the algorithm must have scheduled j in $I(k, \ell, F_u)$ and so, before the t_{u+1}. This proves the first statement in the lemma.

To prove the second statement of lemma, we use induction on u. Suppose the second statement is true for iteration $u-1$. We show that it holds for u. Let j be job of class $k \leq k^*$ such that $j \in \mathcal{I}_{u+1}$ and $r_j^u < t_{u+1}$. Note that interval $I(k, \ell, F_u)$ ends on or after t_{u+1}. Hence $t_{u+1} - r_j^u \leq |I(k, \ell, F_u)| = \frac{T_u \epsilon^k}{\epsilon^3}$. If $r_j \geq t_u$, then $r_j^u = r_j$, and we have $t_{u+1} - r_j \leq |I(k, \ell, F_u)| = \frac{F_u \epsilon^k}{\epsilon^3} = \frac{F_{u+1} \epsilon^k}{(1+\epsilon^3)} \leq F_{u+1} \epsilon^k$.

On the other hand, if $r_j^u = t_u$. So we get $t_{u+1} - t_u \leq |I(k^*, \ell^*, F_u)| \leq \frac{F_u \epsilon^{k^*}}{\epsilon^3} \leq \frac{F_u \epsilon^k}{\epsilon^3}$. By induction hypothesis, we have $t_u - r_j < F_u \epsilon^k$. Hence we have $t_{u+1} - r_j = t_{u+1} - t_u + t_u - r_j \leq \left(\frac{1+\epsilon^3}{\epsilon^3}\right) F_u \epsilon^k = F_{u+1} \epsilon^k$. ◀

▶ **Lemma 12.** *If $\mathcal{C}$ does not finish all jobs in the iteration u, then the value of offline optimal solution is at least F_u.*

Proof. The proof is similar to Lemma 8. The set S is defined similarly. For each machine and interval $I(k, \ell, F_u)$ the algorithm rejects at least ϵ^2-fraction of volume of jobs. Lemma 4 and Lemma 6 remain unchanged.

Note that for a job j of type (k, ℓ, F_u), r_j^u may lie earlier than the start time of $I(k, \ell, F_u)$. So the optimum offline algorithm may complete processing j even before the start of this interval. But Lemma 11 shows that j is released at most $\epsilon^3 |I(k, \ell, F_u)|$ to the left of $I(k, \ell, F_u)$. So in the definition of the intervals $I^\epsilon(k, \ell, F_u)$ in Lemma 4, we consider the interval $I(k, \ell, F_u)$ and two segments of length $\epsilon^3 |I(k, \ell, F_u)|$ both before and after $I(k, \ell, F_u)$. Rest of the arguments are same as in the proof of Lemma 8. ◀

▶ **Corollary 13.** *Suppose OPT lies between F_{u-1} and F_u. Then the algorithm $\mathcal{C}$ completes a job of class k with flow-time at most $\frac{(1+\epsilon^3) F_u \epsilon^k}{\epsilon^3}$*

5.2 Making the algorithm online

We now describe the final on-line algorithm $\mathcal{D}$. The above theorem implies that for any job j, we will know the machine on which it get schedules by time $r_j + \frac{(1+\epsilon^3) F_u \epsilon^k}{\epsilon^3}$. At this time, we place j on the queue of the machine to which it gets scheduled on by $\mathcal{C}$. Further each machine prefer the jobs of larger class and within a particular class, it just goes by processing times. We reject at least $2\epsilon^2$ volume of jobs in each interval. To achieve this, the algorithm rejects job $4\epsilon^2$-jobs (ϵ-fraction as in $\mathcal{A}$ and 3ϵ-fraction on each machine i) in the description of the algorithm $\mathcal{B}$. Hence Property 2 for the algorithm $\mathcal{B}$ can be changed slightly to show that $|P(i, k, \ell)|$ is at most $\frac{(1-2\epsilon^3) F \epsilon^k}{\epsilon^3}$.

▶ **Theorem 14.** *In the schedule $\mathcal{D}$ a job j of class k has flow-time at most $\frac{F \epsilon^k}{\epsilon^8}$. Hence the algorithm $\mathcal{D}$ is an $O(\frac{1}{\epsilon^9})$-competitive algorithm that rejects at most $O(\epsilon)$-fraction of total weights of job.*

References

1 C. Ambühl and M. Mastrolilli. On-line scheduling to minimize max flow time: an optimal preemptive algorithm. *Oper. Res. Lett.*, 33(6):597–602, 2005.

2 S. Anand, K. Bringmann, T. Friedrich, N. Garg, and A. Kumar. Minimizing Maximum (Weighted) Flow-time on Related and Unrelated Machines. In *Proceedings of International Colloquium on Automata, Languages and Programming*, pages 13–24, 2013.

3 S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of Symposium on Discrete Algorithms*, pages 1228–1241, 2012.

4 N. Bansal and E. Cloostermans. Minimizing Maximum Flow-Time on Related Machines. In *Proceedings of Workshop on Approximation Algorithms for Combinatorial Problems*, pages 1–14, 2015.

5 N. Bansal and K. Pruhs. Server Scheduling in the Weighted ℓp Norm. In *Proceedings of Latin American Symposium on Theoretical Informatics*, pages 434–443, 2004.

6 Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Online Scheduling to Minimize Maximum Response Time and Maximum Delay Factor. *Theory of Computing*, 8(1):165–195, 2012.

7 A.R. Choudhury, S. Das, N. Garg, and A .Kumar. Rejecting jobs to Minimize Load and Maximum Flow-time. In *Proceedings of Symposium on Discrete Algorithms*, pages 1114–1133, 2015.

8 A.R. Choudhury, S. Das, A. Kumar, P. Harsha, and G. Ramalingam. Minimizing weighted ℓ_p-norm of flow-time in the rejection model. In *Proceedings on the Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 45, pages 25–37, 2015.

9 P.F. Dutot, E. Saule, A. Srivastav, and Denis D. Trystram. Online Non-preemptive Scheduling to Optimize Max Stretch on a Single Machine. In *Proceedings of nternational Conference on Computing and Combinatorics*, pages 483–495, 2016.

10 K. Fox, S. Im, and B. Moseley. Energy efficient scheduling of parallelizable jobs. In *Proceedings of Symposium on Discrete Algorithms*, pages 948–957, 2013.

11 A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *Proceedings of Symposium on Discrete Algorithms*, pages 1242–1253, 2012.

12 B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of ACM*, 47(4):617–643, 2000.

13 Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online Non-preemptive Scheduling in a Resource Augmentation Model based on Duality. In *European Symposium on Algorithms (ESA, 2016)*, volume 57, pages 1–17, 2016.

14 B.A. Michael, S. Chakrabarti, and S. Muthukrishnan. Flow and Stretch Metrics for Scheduling Continuous Job Streams. In *Proceedings of the Annual Symposium on Discrete Algorithms*, pages 270–279, 1998.

15 C.A. Phillips, C. Stein, and E. Torng andJ. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

16 Im S, B. Moseley, K. Pruhs, and C. Stein. Minimizing Maximum Flow Time on Related Machines via Dynamic Posted Pricing. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 51:1–51:10, 2017.

17 N.K. Thang. Lagrangian Duality in Online Scheduling with Resource Augmentation and Speed Scaling. In *Proceedings of European Symposium on Algorithms*, pages 755–766, 2013.

On the $\mathsf{AC}^0[\oplus]$ Complexity of Andreev's Problem

Aditya Potukuchi
Department of Computer Science, Rutgers University, USA
http://paul.rutgers.edu/~ap1311/
aditya.potukuchi@cs.rutgers.edu

Abstract

Andreev's Problem is the following: Given an integer d and a subset of $S \subset \mathbb{F}_q \times \mathbb{F}_q$, is there a polynomial $y = p(x)$ of degree at most d such that for every $a \in \mathbb{F}_q$, $(a, p(a)) \in S$? We show an $\mathrm{AC}^0[\oplus]$ lower bound for this problem.

This problem appears to be similar to the list recovery problem for degree-d Reed-Solomon codes over $\mathbb{F}_q$ which states the following: Given subsets $A_1, \ldots, A_q$ of $\mathbb{F}_q$, output all (if any) the Reed-Solomon codewords contained in $A_1 \times \cdots \times A_q$. In particular, we study this problem when the lists $A_1, \ldots, A_q$ are randomly chosen, and are of a certain size. This may be of independent interest.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases List Recovery, Sharp Threshold, Fourier Analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.25

Related Version A full version of the paper is available at https://arxiv.org/abs/1907.07969.

Funding *Aditya Potukuchi*: Research supported in part by NSF grant CCF-1514164.

Acknowledgements I would like to thank Swastik Kopparty for the discussions that led to Section 5, and Bhargav Narayanan for the discussions that led to Theorem 8.

1 Introduction

For a prime power q, let us denote by $\mathbb{F}_q$, the finite field of order q. Let us denote the elements of $\mathbb{F}_q = \{a_1, \ldots, a_q\}$. One can think of $a_1, \ldots, a_q$ as some ordering of the elements of $\mathbb{F}_q$. Let $\mathcal{P}_d = \mathcal{P}_d^q$ be the set of all univariate polynomials of degree at most d over $\mathbb{F}_q$. Let us define the problem which will be the main focus of this paper:

Input: A subset $S \subseteq \mathbb{F}_q^2$, and integer d.
Output: Is there a $p \in \mathcal{P}_q^d$ such that $\{(a_i, p(a_i)) \mid i \in [q]\} \subseteq S$?

The problem of proving NP-hardness of the above function seems to have been first asked in [11]. It was called "Andreev's Problem" and still remains open. One may observe that above problem is closely related to the *List Recovery of Reed-Solomon codes.* In order to continue the discussion, we first define Reed-Solomon codes:

▶ **Definition 1** (Reed-Solomon code). *The degree d Reed-Solomon over $\mathbb{F}_q$, abbreviated as* $\mathrm{RS}[q, d]$ *is the following set:*

$$\mathrm{RS}[q, d] = \{(p(a_1), \ldots, p(a_q)) \mid p \in \mathcal{P}_d^q\}$$

Reed-Solomon codes are one of the most widely (if not the most widely) studied families of error-correcting codes. It can be checked that $\mathrm{RS}[q, d]$ is a $(d+1)$-dimensional subspace of $\mathbb{F}_q^q$ such that every non-zero vector has at least $q - d$ non-zero coordinates. In coding theoretic language, we say that $\mathrm{RS}[q, d]$ is a *linear code* of *block length* q, *dimension* $d+1$ and *distance* $q - d$.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 25; pp. 25:1–25:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *List Recovery* problem for a code $\mathcal{C} \subset \mathbb{F}_q^n$ is defined as follows:

▶ **Definition 2** (List Recovery problem for $\mathcal{C}$).
Input: *Sets* $A_1, \ldots, A_n \subseteq \mathbb{F}_q$.
Output: $\mathcal{C} \cap (A_1 \times \cdots \times A_n)$.

Given the way we have defined these problems, one can see that Andreev's Problem is essentially proving NP-hardness for the List Recovery of Reed-Solomon codes where one just has to output a Boolean answer to the question

$$\mathcal{C} \cap (A_1 \times \cdots \times A_n) \neq \emptyset?$$

Indeed, let us consider a List Recovery instance where the code $\mathcal{C}$ is $\mathrm{RS}[q, d]$, and the input sets are given by $A_1, \ldots, A_q$. Let us identify $(A_1, \ldots, A_q)$ with the set

$$S = \bigcup_{i \in [q]} \{(a_i, z) \mid z \in A_i\} \subseteq \mathbb{F}_q^2$$

and let us identify every codeword $w = (w_1, \ldots, w_q) \in \mathcal{C}$, with a set $w_{\text{set}} = \{(a_i, w_i) \mid i \in [q]\}$. Clearly, we have that $w \in A_1 \times \cdots \times A_q$ if and only if $w_{\text{set}} \subseteq S$. Often, we will drop the subscript on w_{set} and refer to w both as a codeword, and as the set of points it passes through. Further identifying $\mathbb{F}_q^2$ with $[q^2]$, and and parameterizing the problem by $r = \frac{d}{q}$, we view Andreev's Problem as a Boolean function $\mathrm{AP}_r : \{0,1\}^{q \times q} \to \{0,1\}$.

The main challenge here is to prove (or at least conditionally disprove) NP-hardness for Andreev's Problem, which has been open for over 30 years. Another natural problem one could study is the circuit complexity for AP_r. This is the main motivation behind this paper, and we will study the $\mathrm{AC}^0[\oplus]$ complexity of AP_r. We shall eventually see that even this problem needs relatively recent results about the power of $\mathrm{AC}^0[\oplus]$ in our proof. Informally, AC^0 is the class of Boolean functions computable by circuits of constant depth, and polynomial size, using $\wedge$, $\vee$, and $\neg$ gates. $\mathrm{AC}^0[\oplus]$ is the class of Boolean functions computable by circuits of constant depth, and polynomial size, using $\wedge$, $\vee$, $\neg$, and $\oplus$ (MOD_2) gates. The interested and unfamiliar reader is referred to [3] (Chapter 14) for a more formal definition and further motivation behind this class. We show that AP_r cannot be computed by AC^0 circuits for a constant r. This type of result is essentially motivated by a similar trend in the study of the complexity of *Minimum Circuit Size Problem.* Informally, the Minimum Size Circuit Problem (or simply MCSP) takes as input a truth table of a function on m bits, and an integer s. The output is 1 if there is a Boolean circuit that computes the function with the given truth table and has size at most s. It is a *major* open problem to show the NP-hardness of MCSP. A lot of effort has also gone into understanding the circuit complexity of MCSP. Allender et al. [2] proved a superpolynomial AC^0 lower bound, and Hirahara and Santanam [9] proved an almost-quadratic formula lower bound for MCSP. A recent result by Golonev et al. [8] extends [2] and proves an $\mathrm{AC}^0[\oplus]$ lower bound for MCSP. Thus one can seek to answer the same question about AP_r.

We now state our main theorem:

▶ **Theorem 3** (Main Theorem). *For any prime power q, and $r \in (0,1)$, we have that any depth h circuit with $\wedge$, $\vee$, $\neg$, and $\oplus$ gates that computes AP_r on q^2 bits must have size at least* $\exp\left(\tilde{\Omega}\left(hq^{\frac{c^2}{h-1}}\right)\right)$.

We make a remark about the theorem. The most glaring aspect is that $r \in (0,1)$ is more or less a limitation of our proof technique. Of course, as r gets *very* small, i.e., $r = O\left(\frac{1}{q}\right)$, one can find depth 2 circuits of size $q^{O(rq)} = q^{O(1)}$. But, we do not know that the case where, for example, $r = \Theta\left(\frac{1}{\log q}\right)$ is any easier for $\mathrm{AC}^0[\oplus]$.

Related prior work: A result of Bogdanov and Safra

Shortly after this paper was uploaded, Bogdanov pointed out to us that can alternative proof of the $\mathrm{AC}^0[\oplus]$ lower bound can be obtained (using some other theorems that are also used in this paper) from [4] (specifically Theorem 14). As stated, this bound works in the regime $d = Kq$ where $0.89 < K < 1$. One difference in approach is that the lower bound here proceeds via showing that AP_r has a sharp threshold (see Definition 4) whereas in [4], the sharp threshold can be shown for an adversarially restricted version of this problem. We believe that the generality of techniques in this paper could find use elsewhere (for example, such as that in Section 5).

1.1 Results on sharp threshold for AP_c

For a $p \in (0,1)$, let $X_1, X_2, \ldots$ denote independent $\mathrm{Ber}(p)$ random variables. For a family of Boolean functions $f : \{0,1\}^n \to \{0,1\}$, we use $f^{(n)}(p)$ to denote the random variable $f(X_1, \ldots, X_n)$.

▶ **Definition 4** (Sharp threshold). *For a family of monotone functions f, we say that f has a* sharp threshold *at p if for every $\epsilon > 0$, there is an n_0 such that for every $n > n_0$, we have that $\mathbb{P}(f^{(n)}(p(1-\epsilon)) = 0) \geq 0.99$, and $\mathbb{P}(f^{(n)}(p(1+\epsilon)) = 1) \geq 0.99$.*

Henceforth, we shall assume that q is a very large prime power. So, all the high probability events and asymptotics are as q grows. Where there is no ambiguity, we also just use $f(p)$ to mean $f^{(n)}(p)$ and n growing.

One limitation of $\mathrm{AC}^0[\oplus]$ that is exploited when proving lower bounds (including in [8]) for monotone functions is that $\mathrm{AC}^0[\oplus]$ cannot compute functions with "very" sharp thresholds. For a quantitative discussion, let us call the smallest ϵ in the definition above the *threshold interval*. It is known that AC^0 (and therefore, $\mathrm{AC}^0[\oplus]$) can compute (some) functions with threshold interval of $O\left(\frac{1}{\log n}\right)$, for example, consider the following function on Boolean inputs $z_1, \ldots, z_n$: Let $Z_1 \sqcup \cdots \sqcup Z_\ell$ be an equipartition of $[n]$, such that each $|Z_i| \approx \log n - \log\log n$. Consider the function given by

$$f(z_1, \ldots, z_n) = \bigvee_{i \in [\ell]} \left(\bigwedge_{j \in Z_i} z_j \right).$$

This is commonly known as the *tribes* function and is known to have a threshold interval of $O\left(\frac{1}{\log n}\right)$. This is clearly computable by an AC^0 circuit. A construction from [12] gives an AC^0 circuit (in n inputs) of size $n^{O(h)}$ and depth h that has a threshold interval $\tilde{O}\left(\frac{1}{(\log n)^{h-1}}\right)$. A remarkable result from [12] and [6] (Theorem 13 from [6] plus Lemma 3.2 in [1]) says that this is in some sense, tight. Formally,

▶ **Theorem 5** ([12, 6]). *Let n be any integer and $f : \{0,1\}^n \to \{0,1\}$ be a function with threshold interval δ at $\frac{1}{2}$. Any depth h circuit with $\wedge$, $\vee$, $\neg$, and $\oplus$ gates that computes f must have size at least* $\exp\left(\Omega\left(h\,(1/\delta)^{\frac{1}{h-1}}\right)\right)$.

This improves upon previous lower bounds by Shaltiel and Viola [14] who show a size lower bound of $\exp\left((1/\delta)^{\frac{1}{h+2}}\right)$. In [12], this was studied as the *Coin Problem*, which we will also define in Section 4. Given the above theorem, a natural strategy suggests itself. If we could execute the following two steps, then we would be done:

1. Establish Theorem 5 for functions with thresholds at points other than $\frac{1}{2}$.
2. Show that AP_r has a sharp threshold at q^{-r} with a suitably small threshold interval, i.e., $\frac{1}{\operatorname{poly} q}$.

The first fact essentially reduces to approximating p-biased coins by unbiased coins in constant depth. Understanding the second part, naturally leads us to study $\mathrm{AP}_r(p)$ for some $p = p(q)$. Let $A_1, \dots, A_q \subset \mathbb{F}_q$ be independently chosen random subsets where each element is included in A_i with probability p. Let $\mathcal{C}$ be the $\mathrm{RS}[q, rq]$ code. We have $|\mathcal{C}| = q^{rq+1}$. Let us denote

$$X := |(A_1 \times \cdots \times A_q) \cap \mathcal{C}|.$$

For $w \in \mathcal{C}$, let X_w denote the indicator random variable for the event $\{w \in A_1 \times \cdots \times A_q\}$. Clearly, $X = \sum_{w\in\mathcal{C}} X_w$, and for every $w \in \mathcal{C}$, we have $\mathbb{P}(X_w = 1) = p^q$. We first note that for $\epsilon = \omega\left(\frac{\log q}{q}\right)$, and $p = q^{-r}(1-\epsilon)$, we have, using linearity of expectation,

$$\begin{aligned}
\mathbf{E}[X] &= \sum_{w\in\mathcal{C}} \mathbf{E}[X_w] \\
&= |\mathcal{C}| \cdot (q^{-r}(1-\epsilon))^q \\
&= q^{rq+1}\left(q^{-r}(1-\epsilon)\right)^q \\
&= q \cdot (1-\epsilon)^q \\
&\le q \cdot e^{-\epsilon q} \\
&= o(1).
\end{aligned}$$

When $p = q^{-r}(1+\epsilon)$, using a similar calculation as above, we have

$$\mathbf{E}[X] = q \cdot (1+\epsilon)^q \ge q.$$

To summarize, for $\epsilon = \omega\left(\frac{\log q}{q}\right)$, and $p = q^{-r}(1-\epsilon)$, $\mathbf{E}[X] \to 0$, and for $p = q^{-r}(1+\epsilon)$, $\mathbf{E}[X] \to \infty$.

▶ **Lemma 6.** *For $\epsilon = \omega\left(\frac{\log q}{q}\right)$, we have*

$$\mathbb{P}(AP_r(q^{-r}(1-\epsilon)) = 1) \le \exp\left(-\Omega(\epsilon q)\right).$$

Proof. This is just Markov's inequality. We have $\mathbb{P}(\mathrm{AP}_r(p(1-\epsilon)) = 0) = \mathbb{P}(X \ge 1) \le \mathbf{E}[X] \le q \cdot e^{-\epsilon q} = \exp\left(-\Omega(\epsilon q)\right)$. ◀

This counts for half the proof of the sharp threshold for AP_r. The other half forms the main technical contribution of this work. We show the following:

▶ **Theorem 7.** *Let q be a prime power, $r = r(q)$ and $\epsilon = \epsilon(q)$ be real numbers such that $q^{-r} \ge \frac{\log q}{q}$ and $\epsilon = \omega\left(\max\left\{q^{-r}, \sqrt{q^{r-1}\log\left(q^{1-r}\right)}\right\}\right)$.*

Let $A_1, \dots, A_q$ be independently chosen random subsets of $\mathbb{F}_q$ with each point picked independently with probability $q^{-r}(1+\epsilon)$. Then

$$\mathbb{P}((A_1 \times \cdots \times A_q) \cap \mathrm{RS}[q, rq] = \emptyset) = o(1).$$

First, we observe is that when r is bounded away from 0 and 1, then $\epsilon = \frac{1}{\text{poly}(q)}$ suffices. The condition to focus on here is that $q^{-r} \geq \frac{\log q}{q}$. Indeed, one can see that this condition is necessary to ensure that w.h.p, all the A_i's are nonempty. So, for example, if the dimension of $\mathcal{C}$ is $q-1$, then setting $p = q^{-1}(1+\epsilon)$ is enough for $\mathbf{E}[X] = \omega(1)$ but this does not translate to there almost surely being a codeword in $A_1 \times \cdots \times A_q$.

Lemma 6 and Theorem 7 together give us that AP_r has a sharp threshold at $\max\left\{q^{-r}, \frac{\log q}{q}\right\}$ whenever $1 - \frac{1}{q} \geq r \gg \frac{1}{\log p}$. For the sake of completeness one could ask if AP_r has a threshold for all feasible values of r, and we show that the answer is yes. More formally,

▶ **Theorem 8** (Sharp threshold for list recovery). *For every $r = r(q)$, there is a critical $p = p(r, q)$ such that for every $\epsilon > 0$,*

1. $\mathbb{P}\left(AP_r(p(1-\epsilon)) = 1\right) = o(1)$.
2. $\mathbb{P}\left(AP_r(p(1+\epsilon)) = 1\right) = 1 - o(1)$.

The case that is not handled by Theorem 7 is when $r = O\left(\frac{1}{\log q}\right)$ (since in this case, Theorem 7 requires $\epsilon = \Omega(1)$). This corresponds to the case where q^{-r} is a number bounded away from 0 and 1.

1.2 Results on random list recovery with errors

Given a random subset of points in $S \subseteq \mathbb{F}_q^2$, what is the largest fraction of any degree $d = \Theta(q)$ polynomial that is contained in this set? Using the Union Bound, it is easy to see that no polynomial of degree d has more than $d \log_{\frac{1}{p}} q + o(q)$ points contained in S (formal details are given in Section 5). We show that perhaps unsurprisingly, this is the truth. Formally,

▶ **Corollary 9.** *Let S be a randomly chosen subset of $\mathbb{F}_q^2$ where each point is picked independently with probability p. Then with probability $1 - o(1)$,*

$$\max_{w \in \text{RS}[q,d]} |w \cap S| = d \log_{\frac{1}{p}} q - O\left(\frac{q}{\log\left(\frac{1}{p}\right)}\right).$$

We restrict our attention to the case when $d = \Theta(q)$, where the above statement is nontrivial. This is the content of Section 5. However, we believe that the statement should hold for all rates, and error (in general) better than $O\left(\frac{q}{\log q}\right)$.

2 Preliminaries

2.1 Properties of Reed-Solomon codes

We will state some basic facts about Reed-Solomon codes that will be used in the proof. The first one is that the dual vector space of a Reed-Solomon code is also a Reed-Solomon code.

▶ **Fact 10.** *Let $\mathcal{C}$ be a $\text{RS}[q, d]$ code. Then $\mathcal{C}^{\perp} = \text{RS}[q, q-d-1]$.*

For $t \neq 0$, let W_t be the number of codewords of weight t in $\text{RS}[q, d]$. We have the following simple bound.

▶ **Proposition 11.** *We have $W_{q-i} \leq \frac{q^{d+1}}{i!}$.*

Proof. We have that $\mathcal{C}$ is a $d+1$-dimensional subspace of $\mathbb{F}^q$. Add i extra constraints by choosing some set of i coordinates and restricting them to 0. As long as $i < d$, these new constraints strictly reduce the dimension of $\mathcal{C}$. There are exactly $\binom{q}{i}$ ways to choose the coordinates, and the resulting space has dimension $d+1-i$. Therefore, the number of codewords of weight at most $q-i$ is at most $q^{d+1-i} \cdot \binom{q}{i} \cdot \leq \frac{q^{d+1}}{i!}$. ◀

2.1.1 Punctured Reed-Solomon codes

All of the statements above when instead of Reed-Solomon codes, one considers *punctured* Reed-Solomon codes. For a $w = (w_1, \ldots, w_n) \in \mathbb{F}_q^n$, and a set $S \subset [n']$, let us define

$$w|_S = (w_i)_{i \in S}.$$

For a subset $\mathcal{C} \subset \mathbb{F}_q^{n'}$, let us define

$$\mathcal{C}|_S := \{w|_S \mid w \in \mathcal{C}\}$$

We call $\mathrm{RS}[q,d]|_S$ the S-punctured $\mathrm{RS}[q,d]$ code. Let $\mathcal{C}$ denote the $\mathrm{RS}[q,d]|_n$ code. Since the properties we will care about are independent of the specific set S, let is just parametrize this by $|S| =: n$. The following properties hold

1. $\mathcal{C}^\perp = \mathrm{RS}[q, q-d-1]_n$.
2. Let W_i be the number of codewords in $\mathcal{C}$ code of weight i. Then we have

   $$W_{n-i} \leq \frac{q^{k-i} n^i}{i!}.$$

   Both facts can be easily checked.

2.2 Basic probability inequalities

We will use the standard (multiplicative) Chernoff bound for sums of i.i.d. Bernoulli random variables. Let $X_1, \ldots, X_n$ be independent $\mathrm{Ber}(p)$ random variables. Let $X := \sum_{i \in [n]} X_i$and denote $\mu = \mathbf{E}[X] = np$. Then for any $\epsilon \in (0,1)$, we have:

$$\mathbb{P}\left(|X - \mu| \geq \epsilon\mu\right) \leq e^{\frac{\epsilon^2 \mu}{2}}. \tag{1}$$

We also have (a special case of) the Paley-Zygmund inequality, which states that for a nonnegative random variable X,

$$\mathbb{P}(X > 0) \geq \frac{\mathbf{E}^2[X]}{\mathbf{E}[X^2]}. \tag{2}$$

2.3 Fourier analysis over $\mathbb{F}_q$

For functions $u, v : \mathbb{F}_q^n \to \mathbb{C}$, we have a normalized inner product $\langle u, v \rangle := \frac{1}{q^n} \sum_{s \in \mathbb{F}_q^n} u(s)\overline{v(s)}$. Consider any symmetric, non-degenerate bi-linear map $\chi : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{R}/\mathbb{Z}$ (such a map exists). For an $\alpha \in \mathbb{F}_q^n$, the *character* function associated with α, denoted by $\chi_\alpha : \mathbb{F}_q^n \to \mathbb{C}$ is given by $\chi_\alpha(x) = e^{-2\pi i \chi(\alpha, x)}$.

We have that for all distinct $\alpha, \beta \in \mathbb{F}_q$, we have that $\langle \chi_\alpha, \chi_\beta \rangle = 0$, and every function $f : \mathbb{F}_q \to \mathbb{C}$ can be written in a unique way as $f(x) = \sum_{\alpha \in \mathbb{F}_q} \widehat{f}(\alpha)\chi_\alpha(x)$. Here the $\widehat{f}(\alpha)$'s are called the *Fourier coefficients*, given by

$$\widehat{f}(\alpha) = \langle f, \chi_\alpha \rangle.$$

We will state some facts that we will use in the proof of Theorem 7. The interested reader is referred to the excellent book of Tao and Vu [15] (chapter 4) for further details.

▶ **Fact 12** (Plancherel's Theorem). *For functions $f, g : \mathbb{F}^n \to \mathbb{C}$, we have*

$$\langle f, g\rangle = \sum_{\alpha} \hat{f}(\alpha)\hat{g}(\alpha).$$

▶ **Fact 13.** *Suppose $g : \mathbb{F}_q^n \to \mathbb{C}$ can be written as a product $g(x) = \prod_{i\in[t]} g_i(x)$, then we have the Fourier coefficients of g given by:*

$$\begin{aligned}\widehat{g}(\alpha) &= (\widehat{g_1} * \cdots * \widehat{g_t})(\alpha)\\ &= \sum_{\beta_1,\ldots,\beta_{t-1}} \widehat{g_1}(\beta_1)\cdots\widehat{g_{t-1}}(\beta_{t-1})\widehat{g_t}(\alpha - \sum_{i\in[q-1]} \beta_i).\end{aligned}$$

▶ **Fact 14.** *If $g : \mathbb{F}_q^n \to \mathbb{C}$ is the indicator of a linear space $\mathcal{C}$, we have:*

$$\widehat{g}(\alpha) = \begin{cases} \frac{|\mathcal{C}|}{|\mathbb{F}|^n}, & \text{if } \alpha \in \mathcal{C}^{\perp}\\ 0, & \text{otherwise.}\end{cases}$$

2.4 Hypercontractivity and sharp thresholds

Here we state some tools from the analysis of Boolean function that we will use:

▶ **Definition 15.** *We say that a function $f : \{0,1\}^n \to \{0,1\}$ is* transitive-symmetric *if for every $i, j \in [n]$, there is a permutation $\sigma \in \mathfrak{S}_n$ such that:*

1. $\sigma(i) = j$

2. $f(x_{\sigma(1)}, \ldots, x_{\sigma_n}) = f(x)$ for all $x \in \{0,1\}^n$.

Let $f : \{0,1\} \to \{0,1\}$ be a monotone function. We will state an important theorem by Friedgut and Kalai, as stated in the excellent reference [13], regarding sharp thresholds for balanced symmetric monotone Boolean functions. This will be another important tool that we will use.

▶ **Theorem 16** ([7]). *Let $f : \{0,1\}^n \to \{0,1\}$ be a nonconstant, monotone, transitive-symmetric function and let $F : [0,1] \to [0,1]$ be the strictly increasing function defined by $F(p) = \mathbb{P}(f(p) = 1)$. Let p_{crit} be the critical probability such that $F(p_{crit}) = 1/2$ and assume without loss of generality that $p_{crit} \le 1/2$. Fix $0 < \epsilon < 1/4$ and let*

$$\eta = B\log(1/\epsilon)\cdot\frac{\log(1/p_{crit})}{\log n},$$

where $B > 0$ is a universal constant. Then assuming $\eta \le 1/2$,

$$F(p_{crit}\cdot(1-\eta)) \le \epsilon, \quad F(p_{crit}\cdot(1+\eta)) \ge 1-\epsilon.$$

We will use an immediate corollary of the above theorem.

▶ **Corollary 17.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a nonconstant, monotone, transitive-symmetric function. Let $F : [0,1] \to [0,1]$ be the strictly increasing function defined by $F(p) = \Pr(f(p) = 1)$. Let p be such that $F(p) \ge \epsilon$, and let $\eta = B\log(1/\epsilon)\cdot\frac{\log(1/p_c)}{\log n}$. Then $F(p(1+2\eta)) \ge 1-\epsilon$.*

In particular, in the above corollary, if for some $\epsilon \in (0,1)$ we have that $F^{-1}(\epsilon) \in (0,1)$, then the function f has a sharp threshold.

One easy observation that will allow us to use Theorem 16 is the following:

▶ **Proposition 18.** *The Boolean function $AP_r : \{0,1\}^{q\times n} \to \{0,1\}$ is transitive-symmetric.*

Proof. For a pair of coordinates indexed by (i_1, j_1) and (i_2, j_2), it is easy to see that the map $(x, y) \mapsto (x + i_2 - i_1, y + j_2 - j_1)$ gives us what we need since the set of polynomials is invariant under these operations. ◀

3 Proof ideas of Theorem 7 and Theorem 8

Here, we sketch the proofs of the Theorem 7 and Theorem 8, which can be considered the two main technical contributions of this work. Due to space constraints, we do not reproduce the full proof. However, all the essential ideas are contained in Section 3.2 and Sec 3.3. First, we describe why some obvious approaches do not work.

3.1 What doesn't work, and why

One obvious attempt to prove Theorem 7 is to consider the second moment of $X(= |\mathcal{C} \cap (A_1 \times \cdots \times A_q)|)$ and hope that $\mathbf{E}[X^2] = (1 + o(1))\mathbf{E}^2[X]$. Unfortunately, $\mathbf{E}[X^2]$ is too large. Through a very careful calculation using the weight distribution of Reed Solomon codes which we do not attempt to reproduce here, we have $\mathbf{E}[X^2] = \Omega\left(e^{\frac{1}{p}}\mathbf{E}^2[X]\right)$. So in the regime where, for example, $p = q^{-\Omega(1)}$, this approach is unviable.

To understand this (without the aforementioned involved calculation) in an informal manner, let us fix $p = q^{-r}$ for some fixed constant r. Let us identify the tuple of sets $(A_1, \ldots, A_q)$ with the single set $S = \cup_{i\in[q]}\{(a_i, z) \mid z \in A_i\}$. So, we are choosing a random subset $S \subset \mathbb{F}_q^2$ of size $\approx q^{2-r}$. On the other hand, the objects we are looking for, i.e., codewords, have size q. This is much larger than the standard deviation of $|S|$, which is of the order of $q^{1-(r/2)}$. Thus, conditioning on the existence of some codeword $w \subset \mathbb{F}_q^2$, the distribution of S changes significantly. One way to see this is the following: Using standard Chernoff bounds, one can check that the size of S is almost surely $q^{2-r} \pm O\left(q^{1-(r/2)}\log q\right)$. However, conditioned on $w \in A_1 \times \cdots \times A_q$, the size of S is almost surely $q + q^{-r}(q^2 - q) \pm O\left(q^{1-(r/2)}\log q\right)$ (the additional q comes from the points that make up w). This is much larger than before when r is relatively large. On the other hand, the main point behind (successful) applications of the second moment method is that the distribution does not significantly change after such a conditioning.

One possible way to circumvent the above problem is to pick a uniformly random set $S \subset \mathbb{F}_q^2$ of size q^{2-r}, instead of every point independently with probability q^{-r}. This is a closely related distribution, and it is often the case that Theorems in this model are also true in the above "i.i.d." model. This fact can be also be made formal (see, for example [10] Corollary 1.16). Here, when one conditions on the existence of some codeword w, at least $|S|$ does not change. Thus the second moment method is not ruled out right at the start. However, it seems to be much more technically involved and it is unclear if it is possible to obtain the relatively small threshold interval that is required for Theorem 3 in this way.

3.2 Proof sketch of Theorem 7

The key idea in the proof of this theorem is to count the number of polynomials in the "Fourier basis". Let us consider $f : \mathbb{F}_q^q \to \{0, 1\}$ to be the indicator of $\mathcal{C}$. For $i \in [q]$, let $g_i : \mathbb{F}_q \to \{0, 1\}$ denote the indicator of A_i.

For an extremely brief and informal discussion, what we what we want is essentially $\langle f, \prod_{i\in[q]} g_i\rangle$, which, by Plancharel's identity (see Fact 12) is $\sum_\alpha \widehat{f} \cdot \widehat{\prod_i g_i}(\alpha)$. Since $\mathcal{C}$ is a vector space, we have that $\widehat{f}$ is supported on $\mathcal{C}^\perp$. Moreover, $\widehat{g_i}(\alpha_i)$ is much larger when $\alpha_i = 0$ than when $\alpha_i \neq 0$ if A_i is random. This combined with the fact that most points in $\mathcal{C}^\perp$ have large weight, and a bit more Fourier analysis means that the inner product, $\langle f, \prod_i g_i\rangle$ is dominated by $\widehat{f}(0)\prod_{i\in[q]}\widehat{g_i}(0)$ which is the expected number of codewords in $A_1 \times \cdots \times A_q$.

Now we give a slightly less informal sketch.

Proof sketch of Theorem 7. What we are trying to estimate is exactly

$$X = |\mathcal{C} \cap (A_1 \times \cdots \times A_q)| = \sum_{(x_1,\ldots,x_q)\in\mathbb{F}^q} f(x) \left(\prod_{i\in[q]} g_i(x_i)\right).$$

Using Fourier analysis over $\mathbb{F}_q$, one can show that

$$\begin{aligned}\frac{q^q}{|\mathcal{C}|}\cdot X &= \sum_{(\alpha_1,\ldots,\alpha_q)\in\mathcal{C}^\perp} \prod_{i\in[q]} \widehat{g}_i(\alpha_i) \\ &\geq \prod_{i\in[q]} \widehat{g}_i(0) - \left|\sum_{(\alpha_1,\ldots,\alpha_q)\in\mathcal{C}^\perp\setminus\{0\}} \left(\prod_{i\in[q]} \widehat{g}_i(\alpha_i)\right)\right|. \qquad (3)\end{aligned}$$

Using the fact that $\mathcal{C}$ is an $\mathrm{RS}[q, rq]$ code, one has (see Fact 10) that $\mathcal{C}^\perp$ is an $\mathrm{RS}[q, q-rq-1]$ code. What will eventually help in the proof is that the weight distribution of Reed Solomon codes (and so in particular, $\mathcal{C}^\perp$) is well understood.

Now clearly, it suffices to understand the term $\sum_{(\alpha_1,\ldots,\alpha_q)\in\mathcal{C}^\perp}\left(\prod_{i\in[q]}\widehat{g}_i(\alpha_i)\right) =: R$. One way to control $|R|$ is to control $|R|^2 = R\overline{R}$. Here, one can use the fact that the A_i's are randomly and independently chosen to establish cancellation in many terms of $\mathbf{E}[|R|^2]$. More precisely, one can prove that

$$\mathbf{E}[|R|^2] = \sum_{(\alpha_1,\ldots,\alpha_q)\in\mathcal{C}^\perp\setminus\{\bar{0}\}} \prod_{i\in[q]} \mathbf{E}[|\widehat{g}_i(\alpha_i)|^2]. \qquad (4)$$

It is a more or less standard fact that if A_i is a uniformly random set where every element is included with probability p independently, then

$$\mathbf{E}\left[|\widehat{g}_i(0)|^2\right] = p^2 + \frac{p(1-p)}{q} \qquad (5)$$

and

$$\mathbf{E}\left[|\widehat{g}_i(\alpha_i)|^2\right] = \frac{p(1-p)}{q} \text{ for } \alpha_i \neq 0. \qquad (6)$$

This difference, will be the reason why $|R|$ is typically much smaller than $\prod_{i\in[q]}\widehat{g}_i(0)$. To continue, let us rewrite (4) using (5) and (6) as

$$\begin{aligned}\mathbf{E}[|R|^2] &= \sum_{i=0}^{n-1} \sum_{\substack{\alpha\in\mathcal{C}^\perp\setminus\{\bar{0}\} \\ \mathrm{wt}(\alpha)=n-i}} \left(p^2 + \frac{p(1-p)}{q}\right)^i \left(\frac{p(1-p)}{q}\right)^{n-i} \\ &\leq \sum_{i=0}^{n-1} \frac{q^{q-rq}}{i!} \left(p^2 + \frac{p(1-p)}{q}\right)^i \left(\frac{p(1-p)}{q}\right)^{n-i}\end{aligned}$$

The last inequality is using the fact that $\mathcal{C}^\perp$ is a $\mathrm{RS}[q, q-rq-1]$ code (Fact 10) and thus we understand its weight distribution (Proposition 11). This quantity can be shown to be at most $e\cdot(1-p)^q\cdot q^{q-rq}\left(\frac{p}{q}\right)^q e^{\left(\frac{2pq}{1-p}\right)}$ and so Markov's inequality gives that with probability at least $1-\frac{1}{q}$, one has that

$$|R| \leq \left(eq\cdot(1-p)^q\cdot q^{q-rq}\left(\frac{p}{q}\right)^q e^{\left(\frac{2pq}{1-p}\right)}\right)^{\frac{1}{2}}. \qquad (7)$$

On the other hand, using standard large deviation inequalities (1), we have the following:

▷ Claim 19. For q, c as given above, let $\epsilon = \omega\left(\sqrt{q^{r-1}\log\left(q^{1-r}\right)}\right)$, and $p = q^{-r}(1+\epsilon)$. Then with probability $1 - o(1)$, we have:

$$\prod_{i\in[q]} \widehat{g}_i(0) \geq q^{-rq}(1+0.9\epsilon)^{0.9q}. \tag{8}$$

What remains to be checked is that for the given range of parameters, plugging in the estimates from (8) and (7) inside (3) gives us that $X > 0$, which completes the proof. ◀

3.3 Proof sketch of Theorem 8

The starting point of Thoerem 8 is noticing that the only case not covered by Theorem 7 is $p \in (0,1)$ is some fixed constant, or equivalently $r = O\left(\frac{1}{\log q}\right)$.

Proof sketch of Theorem 8. Here we have a somewhat crude weight distribution result for Reed Solomon codes (Proposition 11) to compute the second moment. We first show that $\mathbf{E}[X^2] = O\left(e^{\frac{1}{p}}\mathbf{E}^2[X]\right)$. Using, for example the Paley-Zygmund Inequality (2), this means that $\mathbb{P}(X > 0) \geq \Omega(e^{-\frac{1}{p}})$. Thus we have that $\{X > 0\}$ with at least some (possibly small) constant probability. But what we need is that $\mathbb{P}(X > 0) \geq 0.99$. For this, we now use the fact that AP_r is monotone and transitive-symmetric. So Corollary 17) gives that for $p' = p + O\left(\frac{1}{\log q}\right)$, we have that $\mathbb{P}(\mathrm{AP}_r(p') = 1) \geq 0.99$. ◀

4 $\mathsf{AC}^0[\oplus]$ lower bound for AP_r

We prove the lower bound by showing that AP_r solves a biased version of the *Coin Problem*, and use the lower bounds known for such kinds of functions, obtained by [12], [6].

▶ **Definition 20** ((p, ϵ)-coin problem). *We say that a circuit $C = C^n$ on n inputs solves the (p,ϵ)-coin problem if*

- *For $X_1, \ldots, X_n \sim \mathrm{Ber}(p(1-\epsilon))$,*

  $$\mathbb{P}(C(X_1, \ldots, X_n) = 0) \geq 0.99$$

- *For $X_1, \ldots, X_n \sim \mathrm{Ber}(p(1+\epsilon))$,*

  $$\mathbb{P}(C(X_1, \ldots, X_n) = 1) \geq 0.99$$

The $\left(\frac{1}{2}, \epsilon\right)$-Coin Problem was explicitly introduced in [5]. We shall abbreviate the (p,ϵ)-Coin Croblem on n variables as $\mathrm{CP}^n(p,\epsilon)$. We observe that a function $f : \{0,1\}^n \to \{0,1\}$ *solves* $\mathrm{CP}^n\,(p,\epsilon)$ if it has a sharp threshold at p with threshold interval at most ϵ. The one obstacle we have to overcome in using Theorem 5 is that AP_r has a sharp threshold at $p^{-c} \ll \frac{1}{2}$. However, we will show how to simulate biased Bernoulli r.v's from almost unbiased ones. Let $\mathrm{C}(s,d)$ to denote the class of functions on n variables which have circuits of size $O(s) = O(s(n))$ and depth $d = d(n)$ using $\wedge$, $\vee$, $\neg$, and $\oplus$ gates. Here, we make the following simple observation about the power of $\mathrm{AC}^0[\oplus]$ circuits to solve biased and unbiased ϵ-coin problem. First, we observe that it is possible to simulate a biased coin using an unbiased one.

▶ **Lemma 21.** *Let s be such that $\frac{1}{2^s} \le p \in (0,1)$, and $\epsilon \le \frac{1}{s^K}$ for a large constant K. Then, there is a CNF F_p on $t \le s^2$-variables such that for inputs $X_1 \dots, X_t \in \mathrm{Ber}\left(\frac{1}{2}+\epsilon\right)$,*

$$\mathbb{P}\left(F_p(X_1,\dots,X_t)=1\right) = p(1+\Omega(\epsilon L))$$

and for inputs $X_1 \dots, X_t \in \mathrm{Ber}\left(\frac{1}{2}-\epsilon\right)$,

$$\mathbb{P}\left(F_p(X_1,\dots,X_t)=1\right) = p\left(1+\frac{1}{2^{\Omega(\sqrt{t})}}-\Omega(\epsilon L)\right)$$

where $L = \lfloor \log_2(1/p) \rfloor$.

The idea is essentially that the AND of k unbiased coin is a 2^{-k}-biased coin. However, some extra work has to be done if we want other biases (say, $(0.15)\cdot 2^k$). We give a sketch of the proof

Proof Sketch. Consider the sequence of integers $\{k_i\}_{i\in\mathbb{N}}$ such that for every i, k_i is the largest such that

$$\prod_{j=1}^{i}\left(1-\frac{1}{2^j}\right)^{k_j} \ge p.$$

We make a basic observation:

▶ **Observation 22.** *We have that $k_1 = \lfloor \log_2(1/p) \rfloor \le s$ and for all $j \ge 2$, we have that $k_j \le 3$.*

Let ℓ be the largest such that $k_\ell > 0$ and $\sum_{i\in[\ell]} i\cdot k_i < s^2$. Let $t = \sum_{i\in[\ell]} i\cdot k_i$. Consider the CNF given by

$$C_p = \bigwedge_{j\in[\ell]}\left(\bigwedge_{i\in k_j} C_i^j\right)$$

where the clause C_i^j is an $\vee$ of j independent variables. It remains to check that C_p satisfies the required properties. ◀

This lemma now gives us the following:

▶ **Lemma 23.** *Let $z \in (0,1)$ be a fixed constant. If $CP^n\left(\frac{1}{n^z}, o(\epsilon\log n)\right) \in C^n(s,h)$, then there is a $t \le \log^2 n$ such that $CP^{nt}\left(\frac{1}{2},\epsilon\right) \in C^{nt}(zs\log n, h+2)$.*

Proof. Let C be a circuit for $\mathrm{CP}^n\left(\frac{1}{n^z},\delta\right)$-coin problem. Replace each input variable with the CNF $F_{\left(\frac{1}{n^z}\right)}$ from Lemma 21 on $t = O(\log^4 n)$ independent variables. Call this circuit C′, on tn variables. If the bias of each of these input variables is $\frac{1}{2}+\epsilon$, then the guarantee of Lemma 21 is that output of the and gate is 1 with probability at least $\frac{1}{n^z}(1+\Omega(\epsilon\log n))$. A similar computation gives that if the bias of the inputs are $\left(\frac{1}{2}-\epsilon\right)$, then the bias of the output is at most $\frac{1}{n^z}(1-\Omega(\epsilon\log n))$. Therefore, C′ solves $\mathrm{CP}^{nt}\left(\frac{1}{2},\epsilon\right)$, and has size at most $s\log n$, and depth $h+2$. ◀

Theorem 7 and Lemma 6, together, now give us the following corollary:

▶ **Corollary 24.** *Let q be a large enough prime power. Then AP_r on q^2 inputs solves the (q^{-r},ϵ) coin problem, for $\epsilon = \omega\left(\max\left\{q^{-r}, q^{\frac{r-1}{3}}\right\}\right)$*

As a result, Theorem 5, and Lemma 21, and Lemma 23 together, give us the following bounded depth circuit lower bound for AP$_r$:

▶ **Theorem 3** (Restated). *For any $r \in (0,1)$, and $h \in \mathbb{N}$, we have that*

$$AP_r \notin C\left(\exp\left\{\tilde{\Omega}\left(hq^{\frac{r^2}{h-1}}\right)\right\}, h\right).$$

5 Random list recovery with errors

In this section, we shall again consider Reed-Solomon codes RS$[q, rq]$ where r is some constant between 0 and 1. Let us slightly abuse notation, as before, and think of a codeword $w \in \text{RS}[q, rq]$ corresponding to a polynomial $p(X)$ as the set of all the zeroes of the polynomial $Y = p(X)$. That is, for a codeword $w = (w_1, \ldots, w_q)$ associated with polynomial p, we think of w as a subset $\{(a_i, p(a_i)) \mid i \in [q]\}$ (recall that $\mathbb{F} = \{a_1, \ldots, a_q\}$). For a set of points $S \subset \mathbb{F}_q^2$ and a codeword w we say the *agreement between w and S* to denote the quantity $|w \cap S|$. For a code $\mathcal{C}$, we say that the *agreement between $\mathcal{C}$ and S* to denote $\max_{w \in \mathcal{C}} |m \cap S|$.

We are interested in the following question: For a set $S \subset \mathbb{F}_q^2$. What is the smallest ℓ such that there exists a $w \in \text{RS}[q, rq]$ such that $|w \cap S| \geq q - \ell$? In other words, what is the largest agreement between RS$[q, rq]$ and S? This is (very close to) the list recovery problem for codes *with errors*. Naturally, we seek to answer this question when S is chosen randomly in an i.i.d. fashion with probability p. Theorem 7 gives asymptotically tight bounds in a relatively straightforward way for constant error rate.

One can observe that the only properties about Reed-Solomon codes that was used in Theorem 7 was the weight distribution in the dual space of codewords. However, (see Section 2.1.1) these are also true for punctured Reed-Solomon codes codes. So, an analogus theorem also holds for punctured Reed Solomon codes. Formally,

▶ **Theorem 25.** *Let q, n, d be integers such that q is a prime power and $n = \omega(\log q)$, and $q^{-\frac{d}{n}} \geq \frac{\log n}{q}$ and let $\epsilon = \omega\left(\max\left\{q^{-\frac{d}{n}}\sqrt{q^{1-\frac{d}{n}}\log\left(q^{1-\frac{d}{n}}\right)}\right\}\right)$. Let $\mathcal{C}$ be an $\text{RS}[q, d]|_n$ code.*

Let $A_1, \ldots, A_n$ be independently chosen random subsets of $\mathbb{F}_q$ with each point picked independently with probability $q^{-\frac{d}{n}}(1+\epsilon)$. Then

$$\mathbb{P}((A_1 \times \cdots \times A_n) \cap \mathcal{C} = \emptyset) = o(1).$$

We do not repeat the proof but is it the exact same as that of Theorem 7. Let $\mathcal{E}_a$ denote the event that the agreement between S and RS$[q, rq]$ is a. Union bound gives us that

$$\mathbb{P}(\mathcal{E}_{q-\ell}) \leq \binom{q}{\ell} q^{rq+1} p^{q-\ell}. \tag{9}$$

So if ℓ is such that the RHS of 9 is $o(1)$. Then the agreement is almost surely less than $q - \ell$. For the other direction, we have the following corollary:

▶ **Corollary 26.** *Let $\epsilon \geq \max\left\{10q^{-\frac{d}{q-\ell}}, \sqrt{q^{1-\frac{d}{q-\ell}} \cdot \log q}\right\}$. Let S be a randomly chosen subset of $\mathbb{F}_q^2$ with each point picked independently with probability at least $q^{-\frac{d}{q-\ell}}(1+\epsilon)$, then with probability at least $1 - o(1)$, the agreement between S and $\text{RS}[q, d]$ is at least $q - \ell$.*

Proof. For $i \in [q-\ell]$, let us denote

$$S_i := \{j \mid (i,j) \in S\}.$$

Let us use $S' := S_1 \times \cdots \times S_{q-\ell}$. Let us denote $\mathcal{C} = \mathrm{RS}[q,d]|_{q-\ell}$. Formally, for a codeword $w \in \mathrm{RS}[q,d]$, denote p_w to be the polynomial corresponding to m. We have

$$\mathcal{C} = \{(i, p_w(i)) \mid i \in [q-\ell])\}$$

We observe that the conditions in Theorem 7 hold, so

$$\mathbb{P}(C' \cap S' = \emptyset) = o(1)$$

as desired. ◀

▶ **Corollary 9** (Restated). *Given a random subset $S \subseteq \mathbb{F}_q^2$ where each point is picked with probability p, then with probability at least $1-o(1)$, the largest agreement* $\mathrm{RS}[q,d]$ *with S is* $d\log_{\frac{1}{p}} q - O\left(\frac{q}{\log\left(\frac{1}{p}\right)}\right)$.

Proof. Let a be an integer that denotes the maximum agreement between S and $\mathrm{RS}(q,d)$. Suppose that $a \leq d\log_{\frac{1}{p}} q$, then setting $\ell = q-a$, and noting that the conditions for Corollary 26 are satisfied, we get that with probability at least $1-o(1)$, there is a polynomial that agrees with the set S in the first $q-\ell$ coordinates. On the other hand, if $a \geq d\log_{\frac{1}{p}} q + 4\frac{q}{\log\left(\frac{1}{p}\right)}$, again, setting $\ell = q-a$, Union Bound gives us:

$$\begin{aligned}\mathbb{P}(\mathcal{E}_{q-\ell}) &\leq \sum_{w\in\mathcal{C}} \sum_{\substack{P\subset\mathbb{F}_q\\|P|=q-\ell}} \mathbb{P}(w|_P \subseteq S)\\ &= \binom{q}{\ell} q^{d+1} p^{q-r}\\ &\ll \frac{1}{q}.\end{aligned}$$

And so we have that with probability at least $1-o(1)$, the agreement of $\mathrm{RS}(q,d)$ with S is $d\log_{\frac{1}{p}} q - O\left(\frac{q}{\log\left(\frac{1}{p}\right)}\right)$. ◀

6 Conclusion

We started off by attempting to prove a bounded depth circuit lower bound for Andreev's Problem. This led us into (the decision version of the) random List Recovery of Reed-Solomon codes. Here we show a sharp threshold for a wide range of parameters, with nontrivial threshold intervals in some cases. However, one of the unsatisfactory aspects about Theorem 8 is that it is proved in a relatively "hands-off" way possibly resulting in a suboptimal guarantee on ϵ. The obvious open problem that is the following:

▶ **Open Problem.** *Is Theorem 8 true with a better bound on ϵ?*

If it is true with a much smaller ϵ, it would extend in a straightforward way to the $\mathrm{AC}^0[\oplus]$ lower bound as well. Another point we would like to make is that the only thing stopping us from proving Theorem 8 for general MDS codes is the lack of Proposition 18.

References

1 Rohit Agrawal. Coin Theorems and the Fourier Expansion. *CoRR*, abs/1906.03743, 2019. arXiv:1906.03743.

2 Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from Random Strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006. doi:10.1137/050628994.

3 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

4 Andrej Bogdanov and Muli Safra. Hardness Amplification for Errorless Heuristics. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007).*, pages 418–426, 2007. doi:10.1109/FOCS.2007.25.

5 J. Brody and E. Verbin. The Coin Problem and Pseudorandomness for Branching Programs. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 30–39, October 2010.

6 Eshan Chattopadhyay, Pooya Hatami, Shachar Lovett, and Avishay Tal. Pseudorandom Generators from the Second Fourier Level and Applications to AC0 with Parity Gates. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, pages 22:1–22:15, 2019. doi:10.4230/LIPIcs.ITCS.2019.22.

7 Ehud Friedgut and Gil Kalai. Every Monotone Graph Property Has A Sharp Threshold, 1996.

8 Alexander Golovnev, Rahul Ilango, Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, and Avishay Tal. AC^0[p] Lower Bounds Against MCSP via the Coin Problem. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, pages 66:1–66:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.66.

9 Shuichi Hirahara and Rahul Santhanam. On the Average-case Complexity of MCSP and Its Variants. In *Proceedings of the 32Nd Computational Complexity Conference*, CCC '17, pages 7:1–7:20, 2017.

10 Svante Janson, Tomasz Łuczak, and Andrej Rucinski. *Preliminaries*. John Wiley, New York, 2000.

11 David S Johnson. The NP-completeness Column: An Ongoing Guide. *J. Algorithms*, 7(2):289–305, June 1986.

12 Nutan Limaye, Karteek Sreenivasaiah, Srikanth Srinivasan, Utkarsh Tripathi, and S. Venkitesh. A fixed-depth size-hierarchy theorem for $\mathrm{AC}^0[\oplus]$ via the coin problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 442–453, 2019. doi:10.1145/3313276.3316339.

13 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, New York, NY, USA, 2014.

14 Ronen Shaltiel and Emanuele Viola. Hardness Amplification Proofs Require Majority. *SIAM J. Comput.*, 39(7):3122–3154, July 2010.

15 Terence Tao and Van H. Vu. *Additive Combinatorics*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2006. doi:10.1017/CBO9780511755149.

The Preemptive Resource Allocation Problem

Kanthi Sarpatwar
IBM T. J. Watson Research Center, Yorktown Heights, NY, United States of America
sarpatwa@us.ibm.com

Baruch Schieber
Computer Science Department, New Jersey Institute of Technology, Newark, NJ, United States of America
https://cs.njit.edu/faculty/sbar
baruch.m.schieber@njit.edu

Hadas Shachnai
Computer Science Department, Technion, Haifa, Israel
hadas@cs.technion.ac.il

Abstract

We revisit a classical scheduling model to incorporate modern trends in data center networks and cloud services. Addressing some key challenges in the allocation of shared resources to user requests (jobs) in such settings, we consider the following variants of the classic *resource allocation problem* (RAP). The input to our problems is a set J of jobs and a set M of homogeneous hosts, each has an available amount of some resource. A job is associated with a release time, a due date, a weight and a given length, as well as its resource requirement. A *feasible* schedule is an allocation of the resource to a subset of the jobs, satisfying the job release times/due dates as well as the resource constraints. A crucial distinction between classic RAP and our problems is that we allow preemption and migration of jobs, motivated by virtualization techniques.

We consider two natural objectives: *throughput maximization* (MaxT), which seeks a maximum weight subset of the jobs that can be feasibly scheduled on the hosts in M, and *resource minimization* (MinR), that is finding the minimum number of (homogeneous) hosts needed to feasibly schedule all jobs. Both problems are known to be NP-hard. We first present an $\Omega(1)$-approximation algorithm for MaxT instances where time-windows form a laminar family of intervals. We then extend the algorithm to handle instances with arbitrary time-windows, assuming there is sufficient slack for each job to be completed. For MinR we study a more general setting with d resources and derive an $O(\log d)$-approximation for any fixed $d \geq 1$, under the assumption that time-windows are not too small. This assumption can be removed leading to a slightly worse ratio of $O(\log d \log^* T)$, where T is the maximum due date of any job.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Packing and covering problems; Theory of computation → Scheduling algorithms

Keywords and phrases Machine Scheduling, Resource Allocation, Vector Packing, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.26

Related Version Full Version: https://arxiv.org/abs/1811.07413

1 Introduction

We revisit a classical scheduling model to incorporate modern trends in data center networks and cloud services. The proliferation of virtualization and containerization technologies, along with the advent of increasingly powerful multi-core processors, has made it possible to execute multiple virtual machines (or *jobs*) simultaneously on the same host, as well as to preempt and migrate jobs with relative ease. We address some fundamental problems in the efficient allocation of shared resources such as CPU cores, RAM, or network bandwidth to several competing jobs. These problems are modeled to exploit multi-job execution and facilitate

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 26; pp. 26:1–26:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

preemption and migration, while respecting resource and timing constraints. Typically, the infrastructure service providers are oversubscribed; therefore, the common goals here include admission control of jobs to maximize throughput, or minimizing the additional resource required to process all jobs.

The broad setting considered in this paper is the following. Suppose we are given a set of jobs J that need to be scheduled on a set of identical hosts M, where each host has a limited amount of one or more resources. Each job $j \in J$ has release time r_j, due date d_j, and length p_j, along with a required amount of the resource s_j ($\bar{s}_j$ for multiple resources). A job j can be preempted and migrated across hosts but cannot be processed *simultaneously* on multiple hosts, i.e., at any time a job can be processed by at most one host. However, multiple jobs can be processed by any host at any given time, as long as their combined resource requirement does not exceed the available resource. We consider two commonly occurring objectives, namely, *throughput maximization* and *resource minimization.*

In the *maximum throughput* (MaxT) variant, we are given a set of homogeneous hosts M and a set of jobs J, such that each job j has a profit $w_j > 0$ and attributes (p_j, s_j, r_j, d_j). The goal is to find a subset $S \subseteq J$ of jobs of maximum profit $\sum_{j \in S} w_j$ that can be feasibly scheduled on M. This problem can be viewed as a preemptive variant of the classic *resource allocation problem* (RAP) [26, 11, 8, 5].

In the *resource minimization* (MinR) variant, we assume that each job j has a resource requirement vector $\bar{s}_j \in [0,1]^d$ as one of the attributes, where $d \geq 1$ is the number of available resources. W.l.o.g., we assume that each host has a unit amount of each of the d resources. A schedule that assigns a set of jobs $S_{i,t}$ to a host $i \in M$ at time t is feasible if $\sum_{j \in S_{i,t}} \bar{s}_j \leq \bar{1}^d$. Given a set of jobs J with attributes $(p_j, \bar{s}_j, r_j, d_j)$, we seek a set of (homogeneous) hosts M of minimum cardinality such that all of the jobs can be scheduled feasibly on M. MinR is a generalization of the classic *vector packing* (VP) problem, in which a set of d-dimensional items needs to be feasibly packed into a minimum number of d-dimensional bins of unit size in each dimension, i.e., the vector sum of all the items packed into each bin has to be less than or equal to $\bar{1}^d$. Any instance of VP can be viewed as an instance of MinR with $r_j = 0$, $d_j = 1$ and $p_j = 1$ for job $j \in J$.

Another application of this general scheduling scenario relates to the allocation of space and time to advertisements by online advertisement platforms (such as Google or Facebook). In the *ad placement problem* [14, 18] we are given a schedule length of T time slots and a collection of ads that need to be scheduled within this time frame. The ads must be placed in a rectangular display area. whose contents can change in different time slots. All ads share the same height, which is the height of the display area, but may have different widths. Several ads may be displayed simultaneously (side by side), as long as their combined width does not exceed the width of the display area. In addition, each advertisement specifies a display count (in the range $1, \ldots, T$), which is the number of time slots during which the ad must be displayed. The actual time slots in which the advertisement will be displayed may be chosen arbitrarily by the scheduler, and, in particular, need not be consecutive. Suppose that each advertisement is associated with some positive profit, and the scheduler may accept or reject any given ad. A common objective is to schedule a maximum-profit subset of ads within a display area of given width. Indeed, this problem can be cast as a special case of MaxT with a single host, where all jobs have the same release time and due date.

1.1 Prior Work

The classical problem of preemptively scheduling a set of jobs with attributes $(p_j, s_j = 1, r_j, d_j)$ on a single machine so as to maximize throughput can be cast as a special case of MaxT with a single host, where each job requires all of the available resource. Lawler [24] showed that in this special case MaxT admits a *polynomial time approximation scheme (PTAS)*, and the problem is polynomially solvable for uniform job weights. For multiple hosts (i.e., $m = |M| > 1$), this special case of MaxT ($s_j = 1$ for all $j \in J$) admits a $\frac{1}{6+\varepsilon}$-approximation, for any fixed $\varepsilon > 0$. This follows from a result of Kalyanasundaram and Pruhs [21].

As mentioned earlier, another special case of MaxT was studied in the context of advertisement placement. The *ad placement* problem was introduced by Adler *et al.* [1] and later studied in numerous papers (see, e.g., [14, 18, 15, 23, 22] and the survey in [25]). Freund and Naor [18] presented a $(1/3 - \varepsilon)$-approximation for the maximum profit version, namely, for MaxT with a single host and the same release time and due date for all jobs.

Fox and Korupula [17] studied our preemptive scheduling model, with job attributes (p_j, s_j, r_j, d_j), under another popular objective, namely, minimizing weighted flow-time. Their work differs from ours in two ways: while they focus on the online versions, we consider our problems in an offline setting. Further, as they note, while the throughput and resource minimization objectives are also commonly considered metrics, their techniques only deal with flow-time. In fact, these objectives are fundamentally different, and we need novel algorithms to tackle them.

The non-preemptive variant of MaxT, known as the *resource allocation problem* (RAP), was introduced by Phillips *et al.* [26], and later studied by many authors (see, e.g., [7, 6, 8, 9, 19, 11] and the references therein).[1] Chakaravarthy *et al.* [9] consider a generalization of RAP and obtain a constant approximation based on a primal-dual algorithm. We note that the preemptive versus non-preemptive problems differ quite a bit in their structural properties.

As mentioned above, MinR generalizes the classic vector packing (VP) problem. The first non-trivial $O(\log d)$-approximation algorithm for VP was presented by Chekuri and Khanna [10], for any fixed $d \geq 1$. This ratio was improved by Bansal, Caprara and Sviridenko [3] to a randomized algorithm with asymptotic approximation ratio arbitrarily close to $\ln d + 1$. Bansal, Eliás and Khan [4] recently improved this ratio further to $0.807 + \ln(d+1)$. A "fractional variant" of MinR was considered by Jansen and Porkolab [20], where time was assumed to be continuous. For this problem, in the case of a single host, they obtain a PTAS, by solving a configuration linear program (rounding the LP solution is not necessary because time is continuous in their case).

Resource minimization was considered also in the context of the ad placement problem. In this variant, all ads must be scheduled, and the objective is to minimize the width of the display area required to make this possible. Freund and Naor [18] gave a 2-approximation algorithm for the problem, which was later improved by Dawande *et al.* [15] to 3/2. This implies a 3-approximation for MinR instances with $d = 1$, where all jobs have the same release time and due date. We note that this ratio can be slightly improved, using the property that $s_j \leq 1$ for all $j \in J$. Indeed, we can schedule the jobs to use the resource, such that the total resource requirements at any two time slots differ at most by one. Thus, the total amount of resource required at any time exceeds the optimum, OPT, at most by one unit, implying the jobs can be feasibly scheduled on $2OPT + 1$ hosts.

Another line of work relates to the non-preemptive version of MinR, where $d = 1$ and the requirement of each job is equal to 1 (see, e.g. [13, 12]); thus, at most one job can be scheduled on each host at any time.

[1] RAP is also known as the *bandwidth allocation problem.*

1.2 Contributions and Techniques

For summarizing our results, we need the notion of *slackness*. Denote the time window for processing job $j \in J$ by $\chi_j = [r_j, d_j]$, and let $|\chi_j| = d_j - r_j + 1$ denote the length of the interval. Throughout the discussion, we assume that time windows are large enough, i.e., there is a constant $\lambda \in (0, 1)$, such that $p_j \leq \lambda|\chi_j|$ for any job j. Such an assumption is quite reasonable in scenarios arising in our applications. We call λ the *slackness* parameter of the instance.

For the MaxT problem, we present (in Section 3) an $\Omega(1)$-approximation algorithm. As mentioned above, the non-preemptive version of this problem is the classic RAP. To see the structural differences between the non-preemptive and preemptive versions, we consider their natural linear programming relaxations. In solving RAP it suffices to have a variable x_{jt} for each job j and time slot t, indicating the start of job j at slot t. This allows to apply a natural randomized rounding algorithm, where job j is scheduled to start at time t with probability x_{jt}. On the other hand, in MaxT a job can be preempted; therefore, each job requires multiple indicator variables. Further, these variables must be rounded in an *all-or-nothing* fashion, i.e., either we schedule all parts of a job or none of them. Our approach to handle this situation is to, somewhat counter-intuitively, "dumb down" the linear program by not committing the jobs to a particular schedule; instead, we choose a subset of jobs that satisfy certain knapsack constraints and construct the actual schedule in a subsequent phase.

We first consider a *laminar* variant of the problem, where the time windows for the jobs are chosen from a laminar family of intervals.[2] This setting includes several important special cases, such as (i) all jobs are released at $t = 0$ but have different due dates, or (ii) jobs are released at different times, but all must be completed by a given due date. Recall that $m = |M|$ is the number of hosts. Our result for the laminar case is a $\frac{1}{2} - \lambda\left(\frac{1}{2} + \frac{1}{m}\right)$-approximation algorithm, assuming that the slackness parameter satisfies $\lambda < 1 - \frac{2}{m+2}$. Using a simple transformation of an arbitrary instance to laminar, we obtain a $\frac{1}{8} - \lambda\left(\frac{1}{2} + \frac{1}{m}\right)$-approximation algorithm for general instances, assuming that $\lambda < \frac{1}{4} - \frac{1}{2(m+2)}$. Our results imply that as λ decreases, the approximation ratio approaches $\frac{1}{2}$ and $\frac{1}{8}$ for the laminar and the general case, respectively.

Subsequently, we tighten the slackness assumption further to obtain an $\Omega(1)$-approximation algorithm for any constant slackness $\lambda \in (0, 1)$ for the laminar case and any constant $\lambda \in (0, \frac{1}{4})$ for the general case. In the special case where the weight of the job is equal to its area, we extend an algorithm due to Chen, Hassin and Tzur [11] to obtain an $\Omega(1)$-approximation guarantee for the general case with no assumption on slackness.

Our algorithm for the laminar case relies on a non-trivial combination of a *packing* phase and a *scheduling* phase. While the first phase ensures that the output solution has high profit, the second phase guarantees its feasibility. To facilitate a successful completion of the selected jobs, we formulate a set of conditions that must be satisfied in the packing phase. Both phases make use of the structural properties of a *laminar* family of intervals. In the packing phase, we apply our rounding procedure (for the LP solution) to the tree representation of the intervals.[3] We further use this tree in the scheduling phase, to feasibly assign the resource to the selected jobs in a bottom-up fashion. Our framework for solving MaxT is general, and may therefore find use in other settings of *non-consecutive* resource allocation.

[2] See the formal definition in Section 2.

[3] This procedure bears some similarity to the *pipage* rounding technique of [2].

For the MinR problem, we obtain (in Section 4) an $O(\log d)$-approximation algorithm for any constant $d \geq 1$, under a mild assumption that any job has a window of size $\Omega(d^2 \log d \log T)$, where $T = \max_j d_j$. We show that this assumption can be removed, leading to a slight degradation in the approximation factor to $O(\log d \log^* T)$, where $\log^* T$ is the smallest integer κ such that $\underbrace{\log \log \ldots \log}_{\kappa \text{ times}} T \leq 1$. Our approach builds on a formulation of the problem as a configuration LP, inspired by the works of [3, 16]. However, we quickly deviate from these prior approaches, in order to handle the time-windows and the extra constraints. Our algorithm involves two main phases: *a maximization phase* and *residual phase*. Roughly speaking, a configuration is a subset of jobs that can be feasibly assigned to a host at a given time slot t. For each t, we choose $O(m \log d)$ configurations with probabilities proportional to their LP-values. In this phase, jobs may be allocated the resource only for part of their processing length. In the second phase, we construct a residual instance based on the amount of time each job has been processed. A key challenge is to show that, for any time window χ, the total "area" of jobs left to be scheduled is at most $1/d$ of the original total area. We use this property to solve the residual instance.

2 Preliminaries

We start with some definitions and notation. For our preemptive variants of RAP, we assume w.l.o.g. that each host has a unit amount of each resource. We further assume that time is slotted. We allow non-consecutive allocation of a resource to each job, as well as job migration. Multiple jobs can be assigned to the same machine at a given time but no job can be processed by multiple machines at the same time. Formally, we denote the set of jobs assigned to host $i \in M$ at time t by $S_{i,t}$. We say that job j is *completed* if there are p_j time slots $t \in [r_j, d_j] = \chi_j$ in which j is allocated its required amount of the resource on some host. A job j is completed if $|\{t \in \chi_j : \exists i \in M \text{ such that } j \in S_{i,t}\}| \geq p_j$. Let $T = \max_{j \in J} d_j$ be the latest due date of any job.

In MaxT, each job $j \in J$ has a resource requirement $s_j \in (0, 1]$. An assignment of a subset of jobs $S \subseteq J$ to the hosts in M is feasible if each job $j \in S$ is completed, and for any time slot t and host $i \in M$, $\sum_{j \in S_{i,t}} s_j \leq 1$, i.e., the sum of requirements of all jobs assigned to host i is at most the available resource. For the MinR variant, we assume multiple resources. Thus, each job j has a resource requirement vector $\bar{s}_j \in [0, 1]^d$, for some constant $d \geq 1$. Further, each host has a unit amount of each of the d resources. An assignment of a set of jobs $S_{i,t}$ to a host $i \in M$ at time t is feasible if $\sum_{j \in S_{i,t}} \bar{s}_j \leq \bar{1}^d$.

Let $a_j = s_j p_j$ denote the total resource requirement (or, *area*) of job $j \in J$ and refer to the quantity w_j/a_j as the *density* of job j. Finally, a set of intervals is *laminar* if for any two intervals χ' and χ'', exactly one of the following holds: $\chi' \subseteq \chi''$, $\chi'' \subset \chi'$ or $\chi' \cap \chi'' = \phi$.

3 Throughput Maximization

We first consider the case where $\mathcal{L} = \{\chi_j : j \in J\}$ forms a *laminar* family of intervals. In Section 3.1, we present an $\Omega(1)$-approximation algorithm for the laminar case when $\lambda \in \left(0, 1 - \frac{2}{m+2}\right)$. Following this, we describe (in Section 3.2) our constant approximation for the general case for $\lambda \in \left(0, \frac{1}{4} - \frac{1}{2(m+2)}\right)$. We then show, in Section 3.3, how to tighten the results to any constant slackness parameter (i) $\lambda \in (0, 1)$ in the laminar case (ii) $\lambda \in (0, \frac{1}{4})$ in the general case. As an interesting corollary, we obtain an $\Omega\left(\frac{1}{\log n}\right)$-approximation algorithm for the general MaxT problem with no slackness assumption. Further, we show that in the special case of maximum utilization (i.e., the profit of each job equals its “area”), we obtain an $\Omega(1)$ guarantee with no assumption on the slackness.

3.1 The Laminar Case

Our algorithm proceeds in two phases. While the first phase ensures that the output solution has high profit, the second phase guarantees its feasibility. Specifically, let $\omega \in (0, 1 - \frac{\lambda}{m}]$ be a parameter (to be determined).

In Phase 1, we find a subset of jobs S satisfying a knapsack constraint for each χ. Indeed, any feasible solution guarantees that the total area of jobs within any time-window $\chi \in \mathcal{L}$ is at most $m|\chi|$. Our knapsack constraints further restrict the total area of jobs in χ to some fraction of $m|\chi|$. We adopt an LP-rounding based approach to compute a subset S that is optimal subject to the further restricted knapsack constraints. (We remark that a dynamic programming approach would work as well. However, such an approach would not provide us with any intuition as to how an optimal solution for the further restricted instance compares with the optimal solution of the original instance.)

In Phase 2 we allocate the resource to the jobs in S, by considering separately each host i at a given time slot $t \in [T]$ as a unit-sized bin (i, t) and iteratively assigning each job $j \in S$ to a subset of such available bins, until j has the resource allocated for p_j distinct time slots. An outline of the two phases is given in Algorithm 1.

Algorithm 1 Throughput maximization algorithm outline.

```
Input: Set of jobs $J$, hosts $M$ and a parameter $\omega \in (0, 1 - \frac{\lambda}{m}]$
Output: Subset of jobs $S \subseteq J$ and a feasible assignment of $S$ to the hosts in $M$
 Phase 1: Select a subset $S \subseteq J$, such that for each $\chi \in \mathcal{L}$:
     $\sum_{j \in S : \chi_j \subseteq \chi} a_j \leq (\omega + \frac{\lambda}{m}) m|\chi|$
 Phase 2: Find a feasible allocation of the resource to the jobs in $S$
```

Phase 1: The algorithm starts by finding a subset of jobs $S \subseteq J$ such that for any $\chi \in \mathcal{L}$: $\sum_{j \in S : \chi_j \subseteq \chi} a_j \leq (\omega + \frac{\lambda}{m}) m|\chi|$. We solve the following LP relaxation, in which we impose stricter constraint on the total area of the jobs assigned in each time window χ.

$$\begin{array}{lll} \textbf{LP:} & \text{Maximize} & \sum_{j \in J} w_j x_j \\ & \text{Subject to:} & \sum_{j : \chi_j \subseteq \chi} a_j x_j \leq \omega m |\chi| \quad \forall \chi \in \mathcal{L} \\ & & 0 \leq x_j \leq 1 \quad\quad\quad\quad\quad\;\; \forall j \in J \end{array}$$

Rounding the Fractional Solution: Suppose $\mathbf{x}^* = (x_j^* : j \in J)$ is an optimal fractional solution for the LP. Our goal is to construct an integral solution $\hat{\mathbf{x}} = (\hat{x}_j : j \in J)$. We refer to a job j with $x_j^* \in (0, 1)$ as a *fractional job*, and to the quantity $a_j x_j^*$ as its *fractional area*. W.l.o.g., we may assume that for any interval $\chi \in \mathcal{L}$, there is at most one job j with $\chi_j = \chi$ such that $0 < x_j^* < 1$, i.e., it is fractional. Indeed, if two such jobs exist, then the fractional value of the higher density job (breaking ties arbitrarily) can be increased to obtain a solution no worse than the optimal. Note, however, that there could be fractional jobs j' with $\chi_{j'} \subset \chi$.

We start by setting $\hat{x}_j = x_j^*$ for all $j \in J$. Consider the tree representation of $\mathcal{L}$, which contains a node (also denoted by χ) for each $\chi \in \mathcal{L}$, and an edge between nodes corresponding to χ and χ', where $\chi' \subset \chi$, if there is no interval $\chi'' \in \mathcal{L}$ such that $\chi' \subset \chi'' \subset \chi$.[4] Our

[4] Throughout the discussion we use interchangeably the terms *node* and *interval* when referring to a time-window $\chi \in \mathcal{L}$.

rounding procedure works in a *bottom-up* fashion. As part of this procedure, we label the nodes with one of two possible colors: *gray* and *black*. Initially, all leaf nodes are colored *black*, and all internal nodes are colored *gray*. The procedure terminates when all nodes are colored *black*. A node χ is colored as *black* if the following property holds:

▶ **Property 1.** *For any path $\mathcal{P}(\chi, \chi_l)$ from χ to a leaf χ_l there is at most one fractional job j such that χ_j lies on $\mathcal{P}(\chi, \chi_l)$.*

We note that the property trivially holds for the leaf nodes. Now, consider a *gray* interval χ with children $\chi_1, \chi_2, \ldots, \chi_\nu$, each colored *black*. Note that χ is well defined because leaf intervals are all colored black. If there is no fractional job that has χ as its time-window, Property 1 follows by induction, and we color χ *black*. Assume now that j is a fractional job that has χ as its time-window (i.e., $\chi_j = \chi$). If there is no other fractional job that has its time-window (strictly) contained in χ, Property 1 is trivially satisfied. Therefore, assume that there are other fractional jobs $j_1, j_2, \ldots, j_l$ that have their time-windows (strictly) contained in χ. Now, we decrease the fractional area (i.e., the quantity $a_j\hat{x}_j$) of j by Δ and increase the fractional area of jobs in the set $\{j_1, j_2, \ldots, j_l\}$ by Δ_k for job j_k, such that $\Delta = \sum_{k\in[l]} \Delta_k$. Formally, we set $\hat{x}_j \to \hat{x}_j - \frac{\Delta}{a_j}$ and $\hat{x}_{j_k} \to \hat{x}_{j_k} + \frac{\Delta_k}{a_{j_k}}$. We choose these increments such that either $\hat{x}_j$ becomes 0, or for each $k \in [l]$, $\hat{x}_{j_k}$ becomes 1. Clearly, in both scenarios, Property 1 is satisfied, and we color χ *black*.

When all nodes are colored *black*, we round up the remaining fractional jobs. Namely, for all jobs j such that $\hat{x}_j \in (0, 1)$, we set $\hat{x}_j = 1$. It is important to note that by doing so we may violate the knapsack constraints. However, in Theorem 1, we bound the violation.

▶ **Theorem 1.** *Suppose $\mathcal{I} = (J, M, \mathcal{L})$ is a laminar instance of* MaxT *with optimal profit W and $\forall j \in J$: $p_j \leq \lambda|\chi_j|$. For any $\omega \in (0, 1 - \frac{\lambda}{m}]$, the subset $S = \{j \in J : \hat{x}_j = 1\}$, obtained as above, satisfies $\sum_{j\in S} w_j \geq \omega W$, and for any $\chi \in \mathcal{L}$, $\sum_{j\in S:\chi_j\subseteq\chi} a_j \leq (\omega + \frac{\lambda}{m})m|\chi|$.*

Proof. We first observe that any optimal solution $\mathbf{x}^*$ for the LP satisfies: $\sum_{j\in J} w_j x^*_j \geq \omega W$. Indeed, consider an optimal solution O for the instance $\mathcal{I}$. We can construct a fractional feasible solution $\mathbf{x}'$ for the LP by setting $x'_j = \omega$ if $j \in O$; otherwise, $x'_j = 0$. Clearly, $\mathbf{x}'$ is a feasible solution for the LP with profit ωW.

Consider an integral solution $\hat{\mathbf{x}}$, obtained by applying the rounding procedure on $\mathbf{x}^*$. We first show that $\sum_{j\in J} w_j\hat{x}_j \geq \omega W$. To this end, we prove that $\sum_{j\in J} w_j\hat{x}_j \geq \sum_{j\in J} w_j x^*_j \geq \omega W$. Suppose we decrease the fractional area of a job j by an amount Δ, i.e., we set $\hat{x}_j \leftarrow \hat{x}_j - \frac{\Delta}{a_j}$. By the virtue of our procedure, we must simultaneously increase the fractional area of some subset of jobs F_j, where for each $k \in F_j$ we have $\chi_k \subset \chi_j$. Further, the combined increase in the fractional area of the jobs in F_j is the same Δ. Now, we observe that the density of job j (i.e., $\frac{w_j}{a_j}$) cannot be higher than any of the jobs in F_j. Indeed, if $j' \in F_j$ has density strictly lower than j, then the optimal solution $\mathbf{x}^*$ can be improved by decreasing the fractional area of j' by some ϵ while increasing that of j by the same amount (it is easy to see that no constraint is violated in this process) – a contradiction. Therefore, our rounding procedure will never result in a loss, and $\sum_{j\in J} w_j\hat{x}_j \geq \sum_{j\in J} w_j x^*_j \geq \omega W$.

We now show that, for each $\chi \in \mathcal{L}$, $\sum_{j\in J:\chi_j\subseteq\chi} a_j\hat{x}_j \leq (\omega + \frac{\lambda}{m})m|\chi|$. First, observe that for any *gray* interval χ the total fractional area is *conserved*. This is true because there is no transfer of fractional area from the subtree rooted at χ to a node outside this subtree until χ is colored black. Now, consider an interval χ that is colored *black*. We note that for any job j with $x^*_j = 0$, our algorithm ensures that $\hat{x}_j = 0$, i.e., it creates no new fractional jobs. Consider the vector $\hat{\mathbf{x}}$ when the interval χ is converted from *gray* to *black*. At this stage, we have that the total (fractional) area packed in the subtree rooted at χ is

$V(\chi) \overset{def}{=} \sum_{j \in J : \chi_j \subseteq \chi} a_j \hat{x}_j \leq \omega m|\chi|$. Let $\mathcal{F}(\chi)$ denote the set of all fractional jobs j' that have their time-windows contained in χ (i.e., $\chi_{j'} \subseteq \chi$). We claim that the maximum increase in $V(\chi)$ by the end of the rounding procedure is at most $\sum_{j' \in \mathcal{F}(\chi)} a_{j'}$. This holds since our procedure does not change the variables $\hat{x}_j \in \{0, 1\}$. Thus, the maximum increase in the total area occurs due to rounding all fractional jobs into complete ones, after all nodes are colored black. To complete the proof, we now show that the total area of the fractional jobs in the subtree rooted at χ satisfies $\mathcal{A}[\chi] \overset{def}{=} \sum_{j' \in \mathcal{F}(\chi)} a_{j'} \leq \lambda|\chi|$. We prove this by induction on the level of node χ. Clearly, if χ is a leaf then the claim holds, since there can exist at most one fractional job j in χ, and $a_j \leq p_j \leq \lambda|\chi|$. Suppose that $\{\chi_1, \chi_2, \ldots \chi_l\}$ are the children of χ. If there is a fractional job j with $\chi_j = \chi$ then, by Property 1, there are no other fractional jobs with time-windows contained in χ. Hence, $\mathcal{A}[\chi] = a_j \leq \lambda|\chi|$. Suppose there is no fractional job with $\chi_j = \chi$; then, by the induction hypothesis: $\mathcal{A}[\chi_k] \leq \lambda|\chi_k|$ for all $k \in [l]$. Further, $\sum_{k \in [l]} |\chi_k| \leq |\chi|$ and $\mathcal{A}[\chi] = \sum_{k \in [l]} \mathcal{A}[\chi_k] \leq \sum_{k \in [l]} \lambda|\chi_k| \leq \lambda|\chi|$. ◀

Let O be an optimal solution for $\mathcal{I}$ satisfying: $\forall \chi \in \mathcal{L} \; : \; \sum_{j \in O : \chi_j \subseteq \chi} a_j \leq cm|\chi|$, for some $c \geq 1$. Then it is easy to verify that any optimal solution $\mathbf{x}^*$ for the LP satisfies: $\sum_{j \in J} w_j x^*_j \geq \frac{\omega}{c} W$. Hence, we have

▶ **Corollary 2.** *Suppose $\mathcal{I} = (J, M, \mathcal{L})$ is a laminar instance of* MaxT*, such that $\forall j \in J : p_j \leq \lambda|\chi_j|$. Let $S^+ \subseteq J$ be a subset of jobs of total profit W satisfying $\forall \chi \in \mathcal{L}$: $\sum_{j \in S^+ : \chi_j \subseteq \chi} a_j \leq cm|\chi|$, for some $c \geq 1$. Then, for any $\omega \in (0, 1 - \frac{\lambda}{m}]$, there exists a subset $S \subseteq J$ satisfying $\sum_{j \in S} w_j \geq \frac{\omega}{c} W$, such that $\forall \chi \in \mathcal{L}$, $\sum_{j \in S : \chi_j \subseteq \chi} a_j \leq (\omega + \frac{\lambda}{m}) m|\chi|$.*

Phase 2: In Phase 1 we obtained a subset $S \subseteq J$, such that for each $\chi \in \mathcal{L}$: $\sum_{j \in S : \chi_j \subseteq \chi} a_j \leq (\omega + \frac{\lambda}{m}) m|\chi|$. We now show that it is always possible to find a feasible packing of all jobs in S. We refer to host i at time t as a *bin* (i, t). In the allocation phase we label a *bin* with one of three possible colors: *white*, *gray* or *black*. Initially, all bins are colored *white*. We color a bin (i, t) gray when some job j is assigned to host i at time t and color it black when we decide to assign no more jobs to this bin. Our algorithm works in a bottom-up fashion and marks an interval χ as *done* when it has successfully completed all the jobs j with $\chi_j \subseteq \chi$. Consider an interval χ such that any $\chi' \subset \chi$ has already been marked *done*. Let $j \in S$ be a job with time-window $\chi_j = \chi$, that has not been processed yet. To complete job j, we must pick p_j distinct time slots in χ and assign it to a bin in each slot. Suppose that we have already assigned the job to $p'_j < p_j$ slots so far. Denote by *avail*$(j) \subseteq \chi$ the subset of time slots where j has not been assigned yet. We pick the next slot and bin as shown in Algorithm 2.

▶ **Theorem 3.** *For any $\lambda < 1 - \frac{2}{m+2}$, there exists a $\frac{1}{2} - \lambda \left(\frac{1}{2} + \frac{1}{m}\right)$-approximation algorithm for the laminar* MaxT *problem, assuming that $p_j \leq \lambda|\chi_j|$ for all $j \in J$.*

Proof. Given an instance $\mathcal{I} = (J, M, \mathcal{L})$ and a parameter $\omega \in (0, 1 - \frac{\lambda}{m}]$, let W denote the optimal profit. We apply Theorem 1 to find a subset of jobs $S \subseteq J$ of profit ωW, such that for any $\chi \in \mathcal{L}$: $\sum_{j \in S : \chi_j \subseteq \chi} a_j \leq (\omega + \frac{\lambda}{m}) m|\chi|$. We now show that there is a feasible resource assignment to the jobs in S for $\omega = \frac{1}{2} - \lambda \left(\frac{1}{2} + \frac{1}{m}\right)$. Clearly, this would imply the theorem.

We show that for the above value of ω Algorithm 2 never reports **fail**, i.e., the resource is feasibly allocated to all jobs in S. Assume towards contradiction that Algorithm 2 reports **fail** while assigning job j. Suppose that j was assigned to $p'_j < p_j$ bins before this **fail**. For $t \in \chi = \chi_j$, we say that bin (i, t) is *bad* if either (i, t) is colored gray, or j has been assigned to some bin (i', t) in the same time slot. We first show that the following invariant holds, as long as no job j^+ such that $\chi \subset \chi_{j^+}$ has been allocated the resource: the number of bad bins

Algorithm 2 Resource allocation to job j in a single time slot.

```
if there exists a gray bin $(i,t)$ in $avail(j)$ then
    let $S_{(i,t)}$ be the set of jobs assigned to this bin
    if $\sum_{j' \in S_{(i,t)}} s_{j'} + s_j \leq 1$ then
        assign $j$ to host $i$ at time $t$
    else if there exists a white bin $(i',t')$ in $avail(j)$ then
        assign $j$ to host $i'$ at time $t'$.
        color $(i,t)$ and $(i',t')$ black
        pair up $(i,t) \leftrightarrow (i',t')$
    else
        report fail
    end if
else if there exists a white bin $(i,t)$ in $avail(j)$ then
    assign $j$ to host $i$ at time $t$
    color the bin $(i,t)$ gray
else
    report fail
end if
```

while processing job j is at most $\lambda m|\chi|$. Assuming that the claim is true in each of the child intervals of χ, $\{\chi_1, \chi_2 \ldots, \chi_l\}$, before any job with time window χ is allocated the resource, we have the number of bad bins = number of gray bins is at most $\sum_{k\in[l]} \lambda m|\chi_k| \leq \lambda m|\chi|$. Now, consider the iteration in which we assign j to host i at time t. If (i,t) is a gray bin, then the number of bad bins cannot increase. On the other hand, suppose (i,t) was white before we assign j. If there are no gray bins in χ, then the number of bad bins is at most $mp_j \leq \lambda m|\chi|$. Suppose there exist some gray bins, and consider those bins of the form (i',t') such that job j has not been assigned to any host at time t'. If there are no such bins, then again the number of bad bins is at most $mp_j \leq \lambda m|\chi|$. Otherwise, we must have considered one such gray bin (i',t') and failed to assign j to host i' at time t'. By the virtue of the algorithm, we must have colored both (i,t) and (i',t') black. Thus, the number of bad bins does not increase, and our claim holds. Now, since we pair the black bins $(i,t) \leftrightarrow (i',t')$ only if $\sum_{j\in S_{(i,t)}} s_j + \sum_{j'\in S_{(i',t')}} s_{j'} > 1$, the total number of black bins $< 2(\omega + \frac{\lambda}{m})m|\chi|$. Hence, the total number of bins that are black or bad is $< (\lambda + 2(\omega + \frac{\lambda}{m}))m|\chi|$. Now, setting $\omega = \frac{1}{2} - \lambda\left(\frac{1}{2} + \frac{1}{m}\right)$, there should be at least one bin (i^*, t^*) that is neither black nor bad. But in this case, we could have assigned j to host i^* at time t^*, which is a contradiction to the assumption that the algorithm reports a **fail**. ◀

For convenience, we restate the claim shown in the proof of Theorem 3.

▶ **Corollary 4.** *Let $\mathcal{I} = (J, M, \mathcal{L})$ be a laminar instance where $p_j \leq \lambda|\chi_j|$ $\forall j \in J$, for $\lambda \in (0,1)$. Let $S \subseteq J$ be a subset of jobs, such that for any $\chi \in \mathcal{L}$: $\sum_{j\in S:\chi_j\subseteq\chi} a_j \leq (\omega + \frac{\lambda}{m})m|\chi|$, where $\omega \leq \frac{1}{2} - \lambda\left(\frac{1}{2} + \frac{1}{m}\right)$. Then, there exists a feasible resource assignment to the jobs in S.*

3.2 The General Case

We use a simple transformation of general instances of MaxT into laminar instances and prove an $\Omega(1)$-approximation guarantee. Let $\mathcal{W}$ denote the set of all time-windows for jobs in J, i.e., $\mathcal{W} = \{\chi_j : j \in J\}$. We now construct a laminar set of intervals $\mathcal{L}$ and a mapping $\mathfrak{L} : \mathcal{W} \to \mathcal{L}$. Recall that $T = \max_{j\in J} d_j$. The construction is done via a binary tree $\mathcal{T}$ whose nodes correspond to intervals $[l,r] \subseteq [T]$. The construction is described in Algorithm 3.

Algorithm 3 Transformation into a laminar set.

```
Input: Job set $J$ and $\mathcal{W} = \{\chi_j : j \in J\}$
Output: Laminar set of intervals $\mathcal{L}$ and a mapping $\mathfrak{L} : \mathcal{W} \to \mathcal{L}$
let $[T]$ be the root node of tree $\mathcal{T}$
while $\exists$ a leaf node $[l, r]$ in $\mathcal{T}$ such that $r - l > 1$ do
    add to $\mathcal{T}$ two nodes $[l, \lfloor \frac{l+r}{2} \rfloor]$ and $[\lfloor \frac{l+r}{2} \rfloor + 1, r]$ as the children of $[l, r]$
end while
let $\mathcal{L}$ be the set of intervals corresponding to the nodes of $\mathcal{T}$
For each $\chi \in \mathcal{W}$, let $\mathfrak{L}(\chi) = \chi'$, where $\chi'$ is the largest interval in $\mathcal{L}$ contained in $\chi$, breaking ties by picking the *rightmost* interval.
```

▶ **Lemma 5.** *In Algorithm 3, the following properties hold (Proof in final version):*

1. *For any $j \in J$, $|\chi_j| \leq 4|\mathfrak{L}(\chi_j)|$.*
2. *For $\chi \in \mathcal{L}$, let $\tilde{\chi} = \{t \in \chi_j : j \in J, \mathfrak{L}(\chi_j) = \chi\}$, i.e., the union of all time-windows in $\mathcal{W}$ that are mapped to χ. Then, $|\tilde{\chi}| \leq 4|\chi|$.*

▶ **Theorem 6.** *For any $\lambda < \frac{1}{4} - \frac{1}{2(m+2)}$, there exists a $\frac{1}{8} - \lambda \left(\frac{1}{2} + \frac{1}{m}\right)$-approximation algorithm for* MaxT *, assuming that $p_j \leq \lambda|\chi_j|$ for all $j \in J$.*

Proof. Given an instance $(J, M, \mathcal{W})$ of MaxT with slackness parameter $\lambda \in (0, 1)$, we first use Algorithm 3 to obtain a laminar set of intervals $\mathcal{L}$ and the corresponding mapping $\mathfrak{L} : \mathcal{W} \to \mathcal{L}$. Consider a new laminar instance $(J_\ell = \{j_\ell : j \in J\}, M_\ell = M, \mathcal{L})$, constructed by setting $\chi_{j_\ell} = \mathfrak{L}(\chi_j)$. Note that if $S_\ell \subseteq J_\ell$ is a feasible solution for this new instance, the corresponding set $S = \{j : j_\ell \in S_\ell\}$ is a feasible solution for the original instance. Let λ_ℓ denote the slackness parameter for the new instance. We claim that $\lambda_\ell \leq 4\lambda$. Assume this is not true, i.e., there exists a job j_ℓ, such that $p_{j_\ell} > 4\lambda|\chi_{j_\ell}|$; however, by Lemma 5, we have $p_{j_\ell} = p_j \leq \lambda|\chi_j| \leq 4\lambda|\chi_{j_\ell}|$. A contradiction. Now, suppose $O \subseteq J$ is an optimal solution of total profit W for the original (non-laminar) instance. Consider the corresponding subset of jobs $O_\ell = \{j_\ell : j \in O\}$. By Lemma 5, for any $\chi \in \mathcal{L}$, $|\tilde{\chi}| \leq 4|\chi|$. It follows that, for any $\chi \in \mathcal{L}$, $\sum_{j_\ell \in O_\ell : \chi_{j_\ell} \subseteq \chi} a_{j_\ell} = \sum_{j \in O : \mathfrak{L}(\chi_j) \subseteq \chi} a_j \leq 4m|\chi|$. Now, we use Corollary 2 for the laminar instance, taking $c = 4$, $S^+ = O_\ell$ and $\lambda_\ell \in (0, 1)$. Then, for any $\omega \in (0, 1 - \frac{\lambda_\ell}{m}]$, there exists $S_\ell \subseteq J_\ell$ of total profit $\sum_{j_\ell \in S_\ell} w_j \geq \frac{\omega}{c} W$, such that $\forall \chi \in \mathcal{L}$, $\sum_{j_\ell \in S_\ell : \chi_j \subseteq \chi} a_{j_\ell} \leq (\omega + \frac{\lambda_\ell}{m}) m|\chi|$. By Corollary 4, there is a feasible assignment of the resource to the jobs in S_ℓ for $\omega \leq \frac{1}{2} - \lambda_\ell \left(\frac{1}{2} + \frac{1}{m}\right)$. Taking $\omega = \dfrac{1}{2} - 4\lambda \left(\dfrac{1}{2} + \dfrac{1}{m}\right) \leq \dfrac{1}{2} - \lambda_\ell \left(\dfrac{1}{2} + \dfrac{1}{m}\right)$, we have the approximation ratio $\frac{w}{c} = \frac{1}{8} - 4\lambda \left(\frac{1}{8} + \frac{1}{4m}\right)$, for any $\lambda < \frac{1}{4} - \frac{1}{2(m+2)}$. We now return to the original instance and take for the solution the set $S = \{j : j_\ell \in S_\ell\}$. ◀

3.3 Eliminating the Slackness Requirements

In this section we show that the slackness requirements in Theorems 3 and 6 can be eliminated, while maintaining a constant approximation ratio for MaxT . In particular, for laminar instances, we show below that Algorithm 1 can be used to obtain a polynomial time $\Omega(1)$-approximation for any constant slackness parameter $\lambda \in (0, 1)$. For general MaxT instances, this leads to an $\Omega(1)$-approximation for any constant $\lambda \in (0, \frac{1}{4})$. We also show a polynomial time $\Omega(\frac{1}{\log n})$-approximation algorithm for general MaxT using no assumption on slackness. We use below the next result, for instances with "large" resource requirement. The proof is deferred to the full version.

▶ **Lemma 7.** *For any $\delta \in (0,1)$ there is an $\Omega(\frac{1}{\log(1/\delta)})$-approximation for any instance $\mathcal{I} = (J, M, \mathcal{W})$ of MaxT satisfying $s_j \geq \delta \; \forall \; j \in J$.*

3.3.1 Laminar Instances

Recall that $m = |M|$ is the number of hosts. Given a fixed $\lambda \in (0,1)$, let

$$\alpha = \alpha(m, \lambda) = \frac{\lambda(1-\lambda)}{1 - \lambda + \frac{\lambda}{m}}. \tag{1}$$

In Phase 1 of Algorithm 1, we round the LP solution to obtain a subset of jobs $S \subseteq J$. We first prove the following.

▶ **Lemma 8.** *Let $\lambda \in (0,1)$ be a slackness parameter, and*

$$\omega = (1-\alpha)(1-\lambda) - \frac{\alpha\lambda}{m}, \tag{2}$$

where α is defined in (1). Then, given a laminar instance $\mathcal{I} = (J, M, \mathcal{L})$ satisfying $p_j \leq \lambda|\chi_j|$ and $s_j \leq \alpha$, there is a feasible allocation of the resource to the jobs in S.

Proof. We generate a feasible schedule of the jobs in S proceeding bottom-up in each laminar tree. That is, we start handling job j only once all the jobs ℓ with time windows $\chi_\ell \subset \chi_j$ have been scheduled. Jobs having the same time window are scheduled in an arbitrary order. Let j be the next job, whose time window is $\chi = \chi_j$. We can view the interval χ as a set of $|\chi|$ time slots, each consisting of m unit size bins. We say that a time slot $t \in \chi_j$ is "bad" for job j if there is no space for one processing unit of j (i.e., an "item" of size s_j) in any of the bins in t; else, time slot t is "good". We note that immediately before we start scheduling job j the number of bad time slots for j is at most $\frac{m|\chi|(1-\lambda)(1-\alpha)-a_j}{m(1-s_j)}$. Indeed, by Theorem 1, choosing for ω the value in (2), after rounding the LP solution the total area of jobs $\ell \in S$, such that $\chi_\ell \subseteq \chi_j$, is at most

$$(\omega + \frac{\alpha\lambda}{m})m|\chi| = ((1-\alpha)(1-\lambda) - \frac{\alpha\lambda}{m} + \frac{\alpha\lambda}{m})m|\chi|. \tag{3}$$

In addition, for a time slot t to be "bad" for job j, each bin in t has to be at least $(1-s_j)$-full. Hence, the number of good time slots for j is at least

$$\begin{aligned} |\chi| - \frac{m|\chi|(1-\lambda)(1-\alpha) - a_j}{m(1-s_j)} &= |\chi|(1 - \frac{(1-\lambda)(1-\alpha)}{1-s_j}) + \frac{a_j}{m(1-s_j)} \\ &\geq \frac{p_j}{\lambda}(1 - \frac{(1-\lambda)(1-\alpha)}{1-s_j}) + \frac{a_j}{m(1-s_j)} \geq p_j \end{aligned}$$

The first inequality follows from the fact that $p_j \leq \lambda|\chi_j| = \lambda|\chi|$, and the second inequality holds since $s_j \leq \alpha$. Hence, job j can be feasibly scheduled, for any $j \in S$. ◀

Using Lemmas 7 and 8, we prove our main result.

▶ **Theorem 9.** *For any $m \geq 1$ and constant $\lambda \in (0,1)$, MaxT admits a polynomial time $\Omega(1)$-approximation on any laminar instance $\mathcal{I} = (J, M, \mathcal{L})$ with slackness parameter λ.*

Proof. Given a laminar instance $\mathcal{I}$ satisfying the slackness condition, we handle separately two subsets of jobs.

Subset 1: Jobs j satisfying $s_j \leq \alpha = \alpha(m, \lambda)$, where α is defined in (1). We solve MaxT for these jobs using Algorithm 1, taking the value of ω as in (2). By Theorem 1, the approximation ratio is $\omega = (1-\alpha)(1-\lambda) - \frac{\alpha\lambda}{m} = (1-\lambda)^2$, i.e., we have a constant factor.
Subset 2: For jobs j satisfying $s_j > \alpha$, use Lemma 7 to obtain an $\Omega(\frac{1}{\log(1/\alpha)})$-approximation.

Taking the best among the solutions for the two subsets of jobs, we obtain an $\Omega(1)$-approximation. ◀

3.3.2 The General Case

Recall that, given a general MaxT instance, $(J, M, \mathcal{W})$, with a slackness parameter $\lambda \in (0,1)$, our transformation yields a new laminar instance $(J_\ell = \{j_\ell : j \in J\}, M_\ell = M, \mathcal{L})$ with a slackness parameter $\lambda_\ell \leq 4\lambda$ (see the proof of Theorem 6). Now, define

$$\alpha_\ell = \alpha_\ell(m, \lambda_\ell) = \frac{\lambda_\ell(1-\lambda_\ell)}{1-\lambda_\ell+\frac{\lambda_\ell}{m}}, \tag{4}$$

and set

$$\omega = (1-\alpha_\ell)(1-\lambda_\ell) - \frac{\alpha_\ell\lambda_\ell}{m}. \tag{5}$$

Then, by Lemma 8, we have that any job $j_\ell \in J_\ell$ selected for the solution set S can be assigned the resource (using Algorithm 1).

▶ **Theorem 10.** *For any $m \geq 1$ and constant $\lambda \in (0, \frac{1}{4})$, MaxT admits a polynomial time $\Omega(1)$-approximation on any instance $\mathcal{I} = (J, M, \mathcal{W})$ with slackness parameter λ.*

Proof. Given such an instance $\mathcal{I}$, consider the resulting laminar instance. As before, we handle separately two subsets of jobs.
Subset 1: For jobs $j_\ell \in J_\ell$ satisfying $s_{j_\ell} \leq \alpha_\ell$, where α_ℓ is defined in (4), apply Algorithm 1 with ω value as in (5). Then, the approximation ratio is $\omega = (1-\lambda_\ell)^2 \geq (1-4\lambda)^2$.
Subset 2: For jobs j_ℓ where $s_{j_\ell} > \alpha_\ell$, use Lemma 7 to obtain $\Omega(\frac{1}{\log(1/\alpha_\ell)})$-approximation.

Taking the best among the solutions for the two subsets of jobs, we obtain an $\Omega(1)$-approximation. ◀

Finally, consider a general instance of MaxT . By selecting $\delta = \frac{1}{n}$, we can apply Lemma 7 to obtain an $\Omega(\frac{1}{\log n})$-approximate solution, S_1 for the jobs $j \in J$ of heights $s_j \geq \frac{1}{n}$. Let S_2 be a solution consisting of all jobs j for which $s_j < \frac{1}{n}$. Note that this solution is feasible since $\sum_{j \in S_2} s_j < 1$. Selecting the highest profit solution between S_1 and S_2, we have:

▶ **Corollary 11.** *There is a polynomial time $\Omega(\frac{1}{\log n})$-approximation algorithm for MaxT .*

3.3.3 Maximizing Utilization

Consider instances of MaxT where the profit gained from scheduling job j is $w_j = a_j = s_j p_j$. We give the proof of the following in [27].

▶ **Theorem 12.** *There is a polynomial time $\Omega(1)$-approximation for any instance of MaxT where $w_j = a_j$ for all $j \in J$.*

4 Resource Minimization

In this section, we consider the MinR problem with d resources, where $d \geq 1$ is some constant. We show that the problem admits an $O(\log d)$-approximation algorithm under some mild assumptions on the slack and minimum window size. Further, we show that the latter assumption can be removed with a slight degradation in the approximation guarantee.

Our approach builds on a formulation of the problem as a configuration LP and involves two main phases: *a maximization phase* and *residual phase*. We start by describing the configuration LP that is at the heart of our algorithm. Let $J_t \subseteq J$ denote the set of all jobs j such that $t \in \chi_j$, i.e., j can be allocated resources at time slot t. For any $t \in [T]$ and $S \subseteq J_t$, $C = (S, t)$ is a valid *configuration* on a single host if $\sum_{j \in S} \bar{s}_j \leq \bar{1}^d$, i.e., the jobs in S can be feasibly assigned to a single host at time slot t. Denote the set of all valid configurations at time t by $\mathcal{C}_t$, and by $\mathcal{C}^j$ the set of all valid configurations (S, t), such that S contains job j. Denote by x_C the indicator variable for choosing configuration C and by m the number of hosts needed to schedule all jobs. The fractional relaxation of the integer program formulation of our problem is given below.

$$\begin{array}{lll} \textbf{Primal}: & \text{Minimize} & m \\ & \text{Subject to:} & m - \sum_{C \in \mathcal{C}_t} x_C \geq 0, \qquad \forall t \in [T] \\ & & -\sum_{C \in \mathcal{C}^j \cap \mathcal{C}_t} x_C \geq -1, \quad \forall j \in J,\ t \in [T] \\ & & \sum_{C \in \mathcal{C}^j} x_C \geq p_j, \qquad \forall j \in J \\ & & x_C \geq 0, \qquad \forall C \end{array}$$

The first constraint ensures that we do not pick more than m configurations for each time slot $t \in [T]$. The second constraint guarantees that at most one configuration is chosen for each job j at a given time t. Finally, the last constraint guarantees that each job j is allocated the resource for p_j time slots, i.e., job j is completed. The proof of the following theorem (see [27]) is similar to a result of Fleischer et al. [16], with some differences due to "negative" terms in the objective of the dual program.

▶ **Theorem 13.** *For any $\epsilon > 0$, there is a polynomial time algorithm that yields a $(1+\epsilon)$-approximate solution for the configuration LP.*

Given the objective value m of the approximate LP solution, we choose for each t $O(m \log d)$ configurations with probabilities proportional to their LP-values. In this phase, jobs may be allocated the resource only for part of their processing length. In the second phase, we construct a residual instance based on the amount of time each job has been processed. A key challenge is to show that, for any time window χ, the total "area" of jobs left to be scheduled is at most $1/d$ of the original total area. We use this property to solve the residual instance. The detailed algorithm and its analysis are given in [27].

▶ **Theorem 14.** *Let $(J, \mathcal{W})$ be an instance of* MinR *with slackness parameter $\lambda \in (0, \frac{1}{4})$. Fix an $\epsilon \in (0,1)$. If $|\chi_j| \geq \frac{1}{m}\theta d^2 \log d \log(T\epsilon^{-\frac{1}{2}})\ \forall\ j \in J$, for sufficiently large constant θ, we obtain an $O(\log d)$ approximation guarantee with probability at least $1-\epsilon$.*

▶ **Theorem 15.** *Let $(J, \mathcal{W})$ be an instance of* MinR *with slackness parameter $\lambda \in (0, \frac{1}{4})$. Fix an $\epsilon \in (0,1)$. There is a polynomial time algorithm that yields an $O(\log d \log^* T)$ approximation ratio with probability at least $1-\epsilon$.*

References

1 Micah Adler, Phillip B Gibbons, and Yossi Matias. Scheduling space-sharing for internet advertising. *Journal of Scheduling*, 5(2):103–119, 2002.

2 Alexander A. Ageev and Maxim Sviridenko. Pipage Rounding: A New Method of Constructing Algorithms with Proven Performance Guarantee. *J. Comb. Optim.*, 8(3):307–328, 2004.

3 Nikhil Bansal, Alberto Caprara, and Maxim Sviridenko. A New Approximation Method for Set Covering Problems, with Applications to Multidimensional Bin Packing. *SIAM J. Comput.*, 39(4):1256–1278, 2009.

4 Nikhil Bansal, Marek Eliáš, and Arindam Khan. Improved approximation for vector bin packing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1561–1579, 2016.

5 Nikhil Bansal, Zachary Friggstad, Rohit Khandekar, and Mohammad R Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. *ACM Transactions on Algorithms*, 10(1):1, 2014.

6 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

7 Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Approximating the Throughput of Multiple Machines in Real-Time Scheduling. *SIAM J. Comput.*, 31(2):331–352, 2001.

8 Gruia Călinescu, Amit Chakrabarti, Howard J. Karloff, and Yuval Rabani. An improved approximation algorithm for resource allocation. *ACM Trans. Algorithms*, 7(4):48:1–48:7, 2011.

9 Venkatesan T Chakaravarthy, Anamitra R Choudhury, Shalmoli Gupta, Sambuddha Roy, and Yogish Sabharwal. Improved algorithms for resource allocation under varying capacity. In *European Symposium on Algorithms*, pages 222–234. Springer, 2014.

10 Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM journal on computing*, 33(4):837–851, 2004.

11 Bo Chen, Refael Hassin, and Michal Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34(5):501–507, 2002.

12 Julia Chuzhoy and Paolo Codenotti. Resource minimization job scheduling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 70–83. Springer, 2009.

13 Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 81–90, 2004.

14 Milind Dawande, Subodha Kumar, and Chelliah Sriskandarajah. Performance bounds of algorithms for scheduling advertisements on a web page. *Journal of Scheduling*, 6(4):373–394, 2003.

15 Milind Dawande, Subodha Kumar, and Chelliah Sriskandarajah. Scheduling web advertisements: a note on the minspace problem. *Journal of Scheduling*, 8(1):97–106, 2005.

16 L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight Approximation Algorithms for Maximum Separable Assignment Problems. *Math. Oper. Res.*, 36(3):416–431, 2011.

17 Kyle Fox and Madhukar Korupolu. Weighted flowtime on capacitated machines. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 129–143. SIAM, 2013.

18 Ari Freund and Joseph Naor. Approximating the advertisement placement problem. *Journal of Scheduling*, 7(5):365–374, 2004.

19 Navendu Jain, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. *ACM Transactions on Parallel Computing*, 2(1):3, 2015.

20 Klaus Jansen and Lorant Porkolab. On preemptive resource constrained scheduling: polynomial-time approximation schemes. *Integer Programming and Combinatorial Optimization*, pages 329–349, 2002.

21 Bala Kalyanasundaram and Kirk Pruhs. Eliminating Migration in Multi-processor Scheduling. *J. Algorithms*, 38(1):2–24, 2001.

22 Arshia Kaul, Sugandha Aggarwal, Anshu Gupta, Niraj Dayama, Mohan Krishnamoorthy, and PC Jha. Optimal advertising on a two-dimensional web banner. *International Journal of System Assurance Engineering and Management*, pages 1–6, 2017.

23 Subodha Kumar, Milind Dawande, and Vijay Mookerjee. Optimal scheduling and placement of internet banner advertisements. *IEEE Transactions on Knowledge and Data Engineering*, 19(11), 2007.

24 Eugene L Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26(1):125–133, 1990.

25 Shinjini Pandey, Goutam Dutta, and Harit Joshi. Survey on Revenue Management in Media and Broadcasting. *Interfaces*, 47(3):195–213, 2017.

26 Cynthia A. Phillips, R. N. Uma, and Joel Wein. Off-line admission control for general scheduling problems. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 879–888, 2000.

27 Kanthi K. Sarpatwar, Baruch Schieber, and Hadas Shachnai. The Preemptive Resource Allocation Problem. *CoRR*, abs/1811.07413, 2018. arXiv:1811.07413.

Online and Offline Algorithms for Circuit Switch Scheduling

Roy Schwartz
Technion – Israel Institute of Technology, Haifa, Israel
schwartz@cs.technion.ac.il

Mohit Singh
Georgia Institute of Technology, Atlanta, GA, USA
mohit.singh@isye.gatech.edu

Sina Yazdanbod
Georgia Institute of Technology, Atlanta, GA, USA
syazdanbod@gatech.edu

Abstract

Motivated by the use of high speed circuit switches in large scale data centers, we consider the problem of *circuit switch scheduling*. In this problem we are given demands between pairs of servers and the goal is to schedule at every time step a matching between the servers while maximizing the total satisfied demand over time. The crux of this scheduling problem is that once one shifts from one matching to a different one a fixed delay δ is incurred during which no data can be transmitted.

For the offline version of the problem we present a $(1 - 1/e - \epsilon)$ approximation ratio (for any constant $\epsilon > 0$). Since the natural linear programming relaxation for the problem has an unbounded integrality gap, we adopt a hybrid approach that combines the combinatorial greedy with randomized rounding of a different suitable linear program. For the online version of the problem we present a (bi-criteria) $((e - 1)/(2e - 1) - \epsilon)$-competitive ratio (for any constant $\epsilon > 0$) that exceeds time by an additive factor of $O(\delta/\epsilon)$. We note that no uni-criteria online algorithm is possible. Surprisingly, we obtain the result by reducing the online version to the offline one.

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

Keywords and phrases approximation algorithm, online, matching, scheduling

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.27

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.02800`.

Funding *Roy Schwartz*: Research is supported by NSF-BSF grant 2016742.
Mohit Singh: Research is supported by NSF- AF:1910423 and NSF-AF:1717947.

1 Introduction

In recent years the vast scaling up of data centers is fueled by applications such as cloud computing and large-scale data analytics. Such computational tasks, which are performed in a data center, are distributed in nature and are spread over thousands of servers. Thus, it is no surprise that designing better and efficient switching algorithms is a key ingredient in obtaining better use of networking resources. Recently, several works have focused on high speed optical circuit switches that have moving optical mirrors [6, 10, 28] or wireless circuits [13, 15, 29].

A common feature of many of these new switching models is that at any time the data can be transmitted on any matching between the senders and the receivers. However, once the switching algorithm decides to reconfigure from the current matching to a new different matching. This is due to physical limitations such as the time it takes to rotate mirrors, a fixed delay is incurred. This delay happens before data can be sent along the new reconfigured

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 27; pp. 27:1–27:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

matching. Indeed even if one mirror rotates the delay must be incurred and no data can be sent. This has led to significant study on obtaining good scheduling algorithms that take this delay into account [18, 21, 27]. The cost in switching between matchings makes the problem different when compared to the classical literature on scheduling in crossbar switching [5], which are usually based on Birkhoff von-Neumann decompositions. In this paper, we focus on finding the schedule that sends as much data as possible in a fixed time window. We aim to design simple and efficient offline and online algorithms, with provable guarantees, for the scheduling problem that incorporates switching delays.

In the circuit switch scheduling problem, we are given a traffic demand matrix $D \in \mathbb{R}_+^{|A|\times|B|}$, where A is the set of senders and B is the set of receivers. D_{ij} denotes the amount of data that needs to be sent from sender i to receiver j. The D_{ij}'s can also be seen as weights on the edges of a complete bipartite graph with vertex set $A \cup B$. We are also given a time window W and a switching time $\delta > 0$. At any time, the algorithm must pick a matching M and duration α for which the data is transmitted along the edges of the matching M that still require data to be sent. When the algorithm changes to another matching M' for another duration α', the algorithm must account for δ amount of time for switching between the two matchings. Indeed even if one edge changes in the matching, the delay must be incurred and no data can be sent on any of the matching edges. The total amount of time that data is sent along matchings as well as switching time between the matchings must total no more than W. The objective is to maximize the total demand that is satisfied.

1.1 Our Results and Contributions

Our main contribution in this paper are simple and efficient algorithms for the offline and online variants of the circuit switch scheduling problem. The following theorem summarizes our result for the offline setting which gives the first constant factor approximation algorithm for all instances.

▶ **Theorem 1.** *Given any constant $\epsilon > 0$, there is a polynomial time algorithm that returns a $(1 - 1/e - \epsilon)$-approximation for the circuit switch scheduling problem.*

It was already noted in Bojja et al. [27] that the circuit switch scheduling problem is a special case of maximizing a monotone submodular function given a knapsack constraint. Unfortunately, the above reduction requires a ground set of exponential size where elements in the ground set corresponds to matchings of senders and receivers. Hence, the rich literature on submodular function maximization (such as [24]) cannot be applied. Indeed the main challenge is the presence of exponential number of matchings that define the configurations.

Bojja et al. [27] show that the greedy algorithm can be implemented in polynomial time and give a guarantee under the assumption that all entries of the data matrix are small as compared to the time window. Unfortunately, it is easy to construct examples where the greedy algorithm does not give a guarantee close to $(1 - \frac{1}{e})$ (Refer to the full version of our paper [23] for an example).

A different approach is to formulate a linear programming relaxation and round the fractional solution. Indeed, it is easy to formulate two natural linear programming relaxations to the circuit switch scheduling problem. The first assigns a distribution over matchings for every time, whereas the second picks configurations with the additional knapsack constraint. Unfortunately, both have an unbounded integrality gap (Refer to the full version of our paper [23] for the gap examples). Thus, a different approach must be used.

We adopt a hybrid approach that combines greedy and rounding of a special linear program to prove the above theorem. We first give an improved analysis of the greedy algorithm and show it gives a $(1 - \frac{1}{e} - \epsilon)$ approximation when $\delta < \epsilon \cdot W$. On the other hand, when $\delta \geq \epsilon W$, the optimal solution only contains $\frac{1}{\epsilon}$ different matchings. While it is not possible to even guess these constant number of matchings in the solution, we can enumerate (approximately) the time these unknown matchings are scheduled. We then formulate an assignment linear program that assigns matchings to each of these guessed time slots. Then a simple randomized rounding gives us the desired approximation in this case.

We also consider the online variant of the problem where the data matrix is not known in advance but is revealed over time. We consider a discrete time process where at each time step, we receive a new data matrix that needs to be transmitted in addition to the traffic demand still left from all preceding time steps. Moreover, we can choose a matching to transmit data at any time step with the constraint that whenever we change the matching from the previous step, no data is transmitted for δ steps. Our main contribution is a reduction from the online variant to the offline variant. To the best of our knowledge, such reductions with a minor loss in the guarantee are seldomly found. This results in a bi-criteria algorithm since the online algorithm is allowed a slightly larger time window than the optimum. We remark that such a bi-criteria approximation is necessary and we refer the reader to the full version of our paper [23] for details. The following theorem summarizes the above.

▶ **Theorem 2.** *Given a β-approximation for the offline circuit switch scheduling problem and an integer $k \geq 3$, there exists an algorithm achieving a competitive ratio of $(1 - {}^{2}\!/_{k}) \frac{\beta}{1+(1-{}^{2}\!/_{k})\beta}$ for the online circuit switch scheduling problem which uses a time window of $W + k\delta$ as compared to a time window of W for the optimum.*

Combining Theorem 1 and Theorem 2, we have the following corollary.

▶ **Corollary 3.** *For any constant $\epsilon > 0$, there exists an algorithm achieving a competitive ratio of $\left(\frac{e-1}{2e-1} - \epsilon\right)$ for the online circuit switch scheduling problem which uses a time window of $W + O\left({}^{\delta}\!/_{\epsilon}\right)$ as compared to a time window of W for the optimum.*

We note that the online algorithm in the above corollary runs in polynomial time. If one is not interested in the running time of the algorithm, but rather interested only in coping with an unknown future, then Theorem 2 gives an online algorithm whose competitive ratio is $({}^{1}\!/_{2} - \epsilon)$ for any arbitrarily small constant $\epsilon > 0$ (by assuming that the offline problem can be solved optimally, i.e., $\beta = 1$).

1.2 Related Work

Bojja et al. [27] were the first to formally introduce the offline variant of the circuit switch scheduling problem. They focused on the special case that all entries of the data matrix are significantly small, and analyzed the greedy algorithm. Though it is known that the greedy algorithm does not provide any worst-case approximation guarantee for the general case of maximizing a monotone submodular function given a knapsack constraint, [27] proved that in the special case of small demand values, where $D_{ij} \leq \epsilon W$ for all i, j they obtain a $\left(1 - \frac{1}{e^{1-\epsilon}}\right)$-approximation. To the best of our knowledge, our algorithm gives the best provable bound for the offline variant of the circuit switch scheduling problem. A different related variant of the problem is when data does not have to reach its destination in one step, i.e., data can go through several different servers until it reaches its destination [18, 21, 27].

A dual approach, given by Liu et al. [20], aims to minimize the total needed time to transmit the entire demand matrix. Since our algorithm aims to maximize the transmitted data in a time window of W, one can use our algorithm as a black box while optimizing over W. It was proven in [19] that the problem of minimizing the time needed to send all of the data is NP-Complete. Hence, we conclude that the circuit switch scheduling problem is also NP-Complete.

The problem of decomposing a demand matrix into matchings, i.e., the decomposition of a matrix into permutation matrices, was considered by [3, 8, 17, 22]. The special cases of zero delay [14] and infinite delay [25] have also been considered. Several related, but slightly different, settings include [7, 11, 26].

Regarding the theoretical problem of maximizing a monotone submodular function given a knapsack constraint, Sviridenko [24] (building upon the work of Khuller et al. [16]) presented a tight $(1 - 1/e)$-approximation algorithm. This tight algorithm enumerates over all subsets of elements of size at most three, and greedily extends each subset of size three, and returns the best solution found. Deviating from the above combinatorial approach of [16, 24], Badanidiyuru and Vondrák [2] and Ene and Nguyen [9] present algorithms that are based on an approach that extrapolates between continuous and discrete techniques. Unfortunately, as previously mentioned, none of the above algorithms can be directly applied to the circuit switch problem due to the size of the ground set.

The online version of the circuit switch scheduling problem has been considered from a queuing theory prospective, with delays [4] and without delays [12]. In these works, guarantees are proven under the assumption that the incoming traffic is from a known distribution or i.i.d. random variables. To the best of our knowledge, the online version has not been studied from a theoretical perspective.

2 Preliminaries

First, let us start with a formal description of the problem. We are given a complete bipartite graph $G = (A, B, E)$ where A and B are the sets of sending and receiving servers, a constant $\delta \geq 0$ and a time window $W \geq 0$. We are also given the traffic demand matrix of the graph, $D \in \mathbb{R}_+^{|A|\times|B|}$, where D_{ij} denotes the amount of data that needs to be sent from sender i to receiver j. The D_{ij}'s can be seen as weights on the edges of the complete bipartite graph. To simplify the notation, for an edge $e = (i, j)$ we abbreviate D_{ij} to D_e. Let $\mathcal{M}$ be the collection of all matchings in G.

▶ **Definition 4.** *The pair (M, α) is called a configuration if $M \in \mathcal{M}$ and $\alpha \in \mathbb{R}_+$.*

The term *scheduling* a configuration (M, α) means sending data via the matching M for a duration of time that equals α. For simplicity of presentation, we also interpret a matching M as a $\{0, 1\}^{|A|\times|B|}$ matrix where $e \in M$ if and only if the entry of edge e in M equals 1. Note that for any edge $e \in M$ the total data sent through e would be $\min(D_e, \alpha)$ and the total amount of data sent by the configuration would be $||\min(D, \alpha M)||_1 = \sum_{e\in M} \min(D_e, \alpha)$ (note that the minimum is taken element-wise). For simplicity of presentation we may use $||.||_1$ and $||.||$ interchangeably.

Switching from a configuration (M, α) to another (M', α') incurs a given constant delay δ, during which no transmission is done. Let $\mathcal{C}$ denote the collection of all possible configurations.

▶ **Definition 5.** *A schedule S of size k is a subset $S \subseteq \mathcal{C}$ such that $|S| = k$. We say that S requires a total time of $\sum_{(M,\alpha)\in S} (\alpha + \delta)$ to be scheduled.*

The total time of the schedule includes both the time for sending data with each configuration and the delay in switching between them. This brings us to the definition of a feasible schedule.

▶ **Definition 6.** *A schedule S is feasible if $\sum_{\alpha:(M,\alpha)\in S}(\alpha + \delta) \leq W$.*

In the offline setting, the goal is to find a feasible schedule S that maximizes the data sent over the given time window of length W. This problem can be formulated as follows:

$$\max\left\{\left|\left|\min\left(D, \sum_{(M,\alpha)\in S}\alpha M\right)\right|\right|_1 : S \subseteq \mathcal{C}, \sum_{\alpha:(M,\alpha)\in S} (\alpha + \delta) \leq W\right\}. \quad (1)$$

We note that $\mathcal{C}$ might be of infinite size. However, we use standard discretization techniques to limit the set of possible values of α in our algorithms. We will discuss this with more detail in the later relevant sections. For now, assume $\mathcal{C}$ is finite. To facilitate the notation and the analysis of our problem, we turn to a well-known class of functions called *submodular functions*.

▶ **Definition 7.** *Given a ground set $N = \{1, 2, 3, ..., n\}$, a set function $f : 2^N \to \mathbb{R}_+$ is a submodular function if for every $A, B \subseteq N$: $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.*

For our problem, define $f : 2^{\mathcal{C}} \to \mathbb{R}_+$ as:

$$f(S) = \left|\left|\min\left(D, \sum_{(M,\alpha)\in S} \alpha M\right)\right|\right|_1 .$$

Moreover, we denote by $f_S((M,\alpha)) = f(S \cup (M,\alpha)) - f(S)$ the marginal gain of the schedule S if the configuration (M,α) was added to it. It has been shown that f is submodular (refer to Theorem 1 in [27]). For the sake of completeness, we state the theorem. Note that f is *monotone* if for every $A \subseteq B \subseteq N$: $f(A) \leq f(B)$.

▶ **Theorem 8** (Theorem 1 in [27]). *The function f is a monotone submodular function.*

For the online version of the problem, we use a discrete time model. Unlike the offline version, in the online setting we do not know the entire traffic matrix of the graph in the beginning. We start with D_0 as the demand matrix already present in the initial graph. At time t an additional traffic matrix D_t is revealed to the algorithm that includes new demands for data that need to be transmitted. In the online version of the problem sending configuration (M,α) means that for the next $\alpha \in \mathbb{Z}_+$ time steps our algorithm is busy sending the matching M. Switching a configuration to a different one incurs an additional delay of $\delta \in \mathbb{N}$ steps, during which no data can be sent. The incoming traffic matrices, at every step starting with the sending of (M,α) and ending with the switching cost (a total of $\alpha + \delta$ time steps), will accumulate and be added to the remaining traffic matrix of the graph.

3 Offline Circuit Switch Scheduling Problem

In this section, we prove Theorem 1 by giving an approximation algorithm for the circuit switch scheduling problem. Our algorithm is a combination of the greedy algorithm as well as a linear programming based approach. We first show that the greedy algorithm gives close to a $(1 - \frac{1}{e})$-approximation if δ, the switching time, is much smaller than the time window. This is done in Section 3.1. In Section 3.2, we give a randomized rounding algorithm for a linear programming relaxation that gives a $(1 - \frac{1}{e})$-approximation but runs in time exponential in number of matchings used in the optimal solution. While the natural linear program for the problem has unbounded gap, we show how to bypass this when the schedule has a constant number of matchings.

3.1 Greedy Algorithm

The greedy algorithm is as follows: at each step choose the configuration that maximizes the amount of data it sends per unit of time it uses. Formally, if R_i is the remaining data demand in the graph after i configurations were already chosen, the greedy algorithm will choose the following configuration to be used next:

$$(M_{i+1}, \alpha_{i+1}) = \text{argmax}_{M\in\mathcal{M},\alpha\in\mathbb{R}_+} \frac{||\min(R_i, \alpha M)||_1}{\alpha + \delta}. \tag{2}$$

Algorithm 1 Greedy Algorithm.

```
1:  Input: $G = (A, B, E), D, \delta, W$
2:  Output: $\{(M_1, \alpha_1), \ldots, (M_r, \alpha_r)\}$
3:  $\mathcal{S} \leftarrow \emptyset$. $i \leftarrow 0$, $R_1 \leftarrow D$.
4:  while $\sum_{\alpha:(M,\alpha)\in\mathcal{S}} (\alpha + \delta) \leq W$ do
5:      $i \leftarrow i + 1$, $(M_i, \alpha_i) \leftarrow \arg\max_{M\in\mathcal{M},\alpha\in\mathbb{R}_+} \frac{||\min(R_i,\alpha M)||}{\alpha+\delta}$.
6:      $\mathcal{S} \leftarrow \mathcal{S} \cup \{(M_i, \alpha_i)\}$, $R_{i+1} \leftarrow R_i - \min(R_i, \alpha_i M_i)$.
7:  end while
8:  $r \leftarrow i$.
9:  if $\sum_{(M,\alpha)\in\mathcal{S}} (\alpha + \delta) > W$ then
10:     $\beta_r \leftarrow W - \delta - \sum_{j=1}^{r-1}(\alpha_j + \delta)$
11:     if $\beta_r \geq 0$ then
12:         $\mathcal{S} \leftarrow (\mathcal{S} \setminus \{(M_r, \alpha_r)\}) \cup \{(M_r, \beta_r)\}$
13:     else
14:         $\mathcal{S} \leftarrow (\mathcal{S} \setminus \{(M_r, \alpha_r)\})$
15:     end if
16: end if
17: return $\mathcal{S}$
```

The greedy algorithm continues to pick configurations until the first time the time constraint is violated or met. Algorithm 1 demonstrates this process. Let r denote this number of steps and $\mathcal{S}_r$ the schedule created after r steps of this algorithm. The last chosen configuration may violate the time window budget and a natural strategy is to reduce its duration to the time window W as is done in Step (11)-(12) of the algorithm. Indeed [27] analyzes this algorithm and shows that it performs well if each entry in data matrix is small. They also show that the above optimization problem can be solved using the maximum weight matching problem. We give a different analysis of the algorithm and show that it gives us a $\left(1 - \frac{1}{e} - \epsilon\right)$-approximation if $\delta < (\frac{e}{2(e-1)}\epsilon) \cdot W$.

▶ **Theorem 9.** *Let $\mathcal{S}_r$ denote the schedule as returned by the greedy algorithm and $\mathcal{O}$ denote the optimal schedule. Then*

$$f(\mathcal{S}_r) \geq \left(1 - \frac{2\delta}{W}\right)\left(1 - \frac{1}{e}\right) f(\mathcal{O}).$$

Proof. To analyze the algorithm, we first show that the objective of the optimal schedule of a slightly smaller time window $W - \delta$ is not much smaller than the optimum value of the optimum schedule for time window W in Lemma 10. Indeed, the lemma states that given any schedule for time window W, for example the optimal schedule, there exists a schedule with time window $W - \delta$ of a comparable objective.

▶ **Lemma 10.** *For any schedule $\mathcal{S}$ for a time window of W, there is a schedule $\tilde{\mathcal{S}}$ on a window of $W - \delta$ time such that $f(\tilde{\mathcal{S}}) \geq \left(1 - \frac{2\delta}{W}\right) f(\mathcal{S})$.*

Proof. Let T_{data} be the total time spent sending data and T_{switch} be the total time spent switching between configurations. Thus, $W = T_{\text{data}} + T_{\text{switch}}$. We prove that we can remove δ time from some configuration or we can remove an entire configuration from $\mathcal{S}$ while reducing the objective by no more than $\frac{2\delta}{W}$ fraction of the objective. Consider the two following cases for the given $\mathcal{S}$. If $T_{\text{data}} \geq \frac{W}{2}$, we have $\frac{f(\mathcal{S})}{T_{\text{data}}} \leq \frac{2}{W} f(\mathcal{S})$. Thus there exists a configuration that we can deduct δ time from and at most lose $\frac{2\delta}{W}f(\mathcal{S})$. If $T_{\text{switch}} \geq \frac{W}{2}$. This means the number of configurations is at least $\frac{W}{2\delta}$. Each configuration on average sends $\frac{2\delta}{W}f(\mathcal{S})$ data. Therefore, there is a configuration we can completely remove from our schedule such that total amount of lost data is at most $\frac{2\delta}{W}f(\mathcal{S})$. In both cases we can reduce the time taken by the schedule by at least δ and have a new schedule $\tilde{\mathcal{S}}$ such that $f\left(\tilde{\mathcal{S}}\right) \geq \left(1 - \frac{2\delta}{W}\right) f(\mathcal{S})$. ◀

Let $\mathcal{O}'$ denote the optimal solution with time window $W - \delta$. From Lemma 10, we have $f(\mathcal{O}') \geq \left(1 - \frac{2\delta}{W}\right) f(\mathcal{O})$. In the following lemma, we show that the output of the greedy algorithm is at least a $\left(1 - \frac{1}{e}\right)$-approximation of $f(\mathcal{O}')$. The proof of the lemma follows standard analysis for greedy algorithms for coverage functions, or more generally submodular functions, except at the last step. For the proof refer to the full version of our paper [23]. The proof of Theorem 9 now follows immediately.

▶ **Lemma 11.** *If $\mathcal{O}'$ is the optimum schedule on time window $W - \delta$, then*

$$f(\mathcal{S}_r) \geq (1 - \frac{1}{e}) f(\mathcal{O}').$$ ◀

3.2 Linear Programming Approach for Constant Number of Configurations

In this section, we assume that we want to schedule at most a given constant k number of configurations and prove the following theorem.

▶ **Theorem 12.** *There exists a randomized polynomial time algorithm that given an integer k and an instance of the circuit switch scheduling problem returns a feasible schedule whose objective, in expectation, is at least $(1 - \frac{1}{e} - \epsilon)$ of the optimum solution that uses at most k matchings. Moreover the running time of the algorithm is polynomial in $\frac{n}{\epsilon^k}$.*

Let us denote optimum schedule by $\mathcal{O} = \{(M_1^*, \alpha_1^*), \ldots, (M_k^*, \alpha_k^*)\}$. Note that, without the loss of generality, we can assume that we know what the α_i^*'s are. This can be done by a standard discretization of the possible values. Since, the number of configurations is constant this enumeration will be polynomial in $\frac{1}{\epsilon^k}$ to an accuracy of ϵ. The total data sent by a schedule $\mathcal{S}$ is $f(\mathcal{S}) = ||\min(D, \sum_{(M,\alpha)\in\mathcal{S}} \alpha M)||_1$. However, in this section, it is more beneficial to consider the total data as the sum of total data sent over each edge. We model the total data by $Z = \sum_{e\in E} z_e$, where z_e is the amount of data that was sent through edge e in our graph. In the case of the optimum, $z_e^* = \min(D_e, \sum_{\alpha^*:(M^*,\alpha^*)\in\mathcal{O}:e\in M^*} \alpha^*)$ and $Z^* = \sum_{e\in E} z_e^*$. We can formulate the following integer program for this problem.[1]

$$
\begin{array}{llll}
(\mathcal{P}) \quad \max & \displaystyle\sum_{e\in E} z_e & & (3) \\
\text{s.t.} & \displaystyle\sum_{M\in\mathcal{M}} x_{M,i} \leq 1 & \forall i = 1,\ldots,k & (4) \\
& z_e \leq D_e & \forall e \in E & (5) \\
& \displaystyle z_e \leq \sum_{i=1}^{k} \sum_{M\in\mathcal{M}:e\in M} \alpha_i^* \cdot x_{M,i} & \forall e \in E & (6) \\
& x_{M,i} \in \{0,1\} & \forall e \in E, \forall M \in \mathcal{M}, \forall i = 1,\ldots,k &
\end{array}
$$

[1] The variable $x_{M,i}$ is the fractional indicator for choosing the configuration (M, α_i^*).

Constraints (4) is to ensure that only one matching is considered in every time interval. Constraint (5) and (6) are to model the total data sent. We can relax this integer program to an LP by changing the $x_{M,i} \in \{0,1\}$ to $0 \leq x_{M,i} \leq 1$. The following lemma states that the relaxed linear program is a relaxation of our problem for the constant number of configurations.

▶ **Lemma 13.** *Let Z_{LP} be the value of an optimum solution to the LP, then $Z_{LP} \geq Z^*$*

Proof. If $\mathcal{O} = \{(M_1^*, \alpha_1^*), \ldots, (M_k^*, \alpha_k^*)\}$ is our optimum answer, based on $\mathcal{O}$ we will create a feasible answer to the LP. For every $(M_i^*, \alpha_i^*) \in \mathcal{O}$, we set $x_{M^*,i} = 1$. Clearly, the constraint 4 is satisfied since we picked exactly one matching for every interval. The constraints 5 and 6 are by definition satisfied since $f(\mathcal{O}) = || \min(D, \sum_{(M,\alpha)\in\mathcal{O}} \alpha M)||$ and the constraints are modeling this minimum. This argument shows that the optimum answer is feasible in the LP and since the LP is a maximization problem we can conclude that $Z_{LP} \geq f(\mathcal{O})$. ◀

The LP contains an exponential number of variables, since the number of matchings in the complete graph is exponential in the size of the graph. To be able to solve this program we need to introduce a separation oracle for the dual of this LP. The following program is the dual of our LP.

$$
\begin{aligned}
(\mathcal{D}) \qquad \min \quad & \sum_{i=1}^{k} y_i + \sum_{e\in E} d_e a_e & & (7)\\
s.t. \quad & y_i \geq \alpha_i^* \sum_{e\in M} b_e & \forall M \in \mathcal{M}, \forall i = 1,\ldots,k \quad & (8)\\
& a_e + b_e \geq 1 & \forall e \in E \quad & (9)\\
& a_e \geq 0,\ b_e \geq 0,\ y_i \geq 0 & \forall e \in E, \forall i = 1,\ldots,k \quad &
\end{aligned}
$$

The Lemma 14 states the existence of a separation oracle.

▶ **Lemma 14.** *The dual program $\mathcal{D}$ admits a polynomial time separation oracle.*

Proof. Given a solution $(\{y_i\}_{i=1}^k, \{a_E\}_{e\in E}, \{b_e\}_{e\in E})$ we are required to determine whether it is feasible and if not provide a constraint that is violated. We can easily determine whether all constraints of type (9) are satisfied, and if not provide one that is violated, by a simple enumeration over all edges $e \in E$. The same can be done for constraints of type (8) by enumerating over $i = 1,\ldots,k$ and for each i compute a maximum weight matching in G equipped with $\{b_e\}_{e\in E}$ as edge weights and check whether the maximum weight matching has value at most y_i/α_i^*. If the maximum weight matching exceeds the target value return the constraint that corresponds to i and the maximum weight matching. ◀

Solving the linear program will provide us with a fractional solution $\{x_{M,i}\}_{M\in\mathcal{M},i=1,\ldots,k}$. For any i we have $\sum_{M\in\mathcal{M}} x_{M,i} \leq 1$. This constraint of the LP creates a distribution over the matchings in time interval i. We create a solution to the program $\mathcal{P}$ from the fractional solution by a randomized rounding technique. We pick $M \in \mathcal{M}$ for the time interval i with probability $x_{M,i}$. Note that with probability $1 - \sum_{M\in\mathcal{M}} x_{M,i}$ no matching will be chosen for this time interval. A formal description of this rounding method is provided in Algorithm 2. Let $X_{M,i}$ denote the indicator random variable if matching M is selected for the i^{th} slot. Moreover, let $Y_{e,i}$ denote the random variable that edge e is present in the matching chosen in the i^{th} slot. We have $Y_{e,i} = \sum_{M\in\mathcal{M}:e\in M} X_{M,i}$ for each $e \in E$ and i and $E[Y_{e,i}] = \sum_{M\in\mathcal{M}:e\in M} x_{M,i}$. Moreover, let Z_e denote the random variable that denotes the data sent along edge e. Then we have $Z_e = \min(D_e, \sum_{i=1}^k \alpha_i^* Y_{e,i})$. Observe that the random variables $\{Y_{e,i}\}_{i=1}^k$ are independent.

Algorithm 2 Randomized Rounding.

```
Input: $(k, \{\alpha_i^*\}_{i=1}^k, \{x_{M,i}\}_{M \in \mathcal{M}, i=1,\dots,k})$
Output: $\{(M_i, \alpha_i^*)\}_{i=1}^k$
for $i \leftarrow 1, \dots, k$ do
    choose $M_i$ to be a random matching w.p. $x_{M,i}$ for the interval $i$
end for
return $\{(M_i, \alpha_i^*)\}_{i=1}^k$
```

The following Lemma 15 is implicit in Theorem 4 of Andelman and Mansour [1].

▶ **Lemma 15.** *Let $Y_1, \dots, Y_n$ be independent Bernoulli random variables and let $Z = \min(B, \sum_{i=1}^n b_i Y_i)$ for some non-negative reals $B, b_1, \dots, b_n$. Then we have that $\mathbb{E}[Z] \geq \left(1 - \frac{1}{e}\right) \min\left(B, \mathbb{E}\left[\sum_{i=1}^n b_i Y_i\right]\right)$.*

Applying the above lemma for each e and random variables $\{Y_{e,i}\}_{i=1}^k$, we obtain that

$$\begin{aligned}\mathbb{E}[Z_e] &\geq \left(1 - \frac{1}{e}\right) \min\left(D_e, \mathbb{E}\left[\sum_{i=1}^k \alpha_i^* Y_{e,i}\right]\right) = \left(1 - \frac{1}{e}\right) \min\left(D_e, \sum_{i=1}^k \sum_{M \in \mathcal{M}: e \in M} \alpha_i^* x_{e,i}\right) \\ &\geq \left(1 - \frac{1}{e}\right) \cdot z_e.\end{aligned}$$

Now summing over all edges, Theorem 12 follows. We are now ready to conclude our discussion of the offline variant of the circuit switch scheduling problem and prove Theorem 1.

Proof of Theorem 1. Given $\epsilon > 0$, if $\delta \leq (\frac{e}{2(e-1)}\epsilon)W$ then Theorem 9 gives us a $(1 - \frac{1}{e} - \epsilon)$-approximation. Otherwise, $\frac{2(e-1)}{e}\frac{1}{\epsilon} > \frac{W}{\delta}$ implying that at most $\frac{2(e-1)}{e}\frac{1}{\epsilon}$ configurations can be scheduled. In this case, Theorem 12 will give a $(1 - \frac{1}{e} - \epsilon)$-approximation. ◀

4 Online Circuit Switch Scheduling Problem

In this section, we prove Theorem 2. Recall that in the online setting, we consider a discrete time model[2] where an additional traffic matrix is revealed at every time $t = 1, 2, \dots, T$. At every time step t, a new set of traffic demands arrives and adds to the remaining traffic that has not been sent so far. We assume that the data matrix arriving at each step is integral and thus can be modeled as a multigraph. We denote the incoming traffic matrices as multigraphs $\{E_1, E_2, \dots, E_T\}$ (instead of D_i's to simplify and familiarize the notation) and thus union of any two such graphs is defined by adding the number of copies of edges in the two constituents. Before proving the general theorem, we first consider the case when there is no delay while switching matchings, i.e., $\delta = 0$. Observe that in this case, the offline problem can be solved exactly and we show a $\frac{1}{2}$-competitive algorithm for the online problem. The general reduction builds on this simple case along with the offline algorithm.

[2] We could also consider a continuous time model where data matrices can arrive at any time and the algorithm can choose a matching at any time instant with a switching time δ when no data is sent. Our results apply to this model as well. The discrete model makes the presentation of the results easier.

4.1 Without Configuration Delay

Observe that an online algorithm, in this case, will pick a set of matchings $\{M_1, M_2, \ldots, M_T\}$, instead of a schedule, that covers the maximum number of edges. At each step t, the algorithm picks the maximum matching from the graph formed by the new edges that arrive, E_t, and the remaining edges in the graph from previous steps which we denote by R_{t-1}. The algorithm is formally given in Algorithm 3. Here $\mathcal{M}$ denotes the set of all matchings on the complete bipartite graph with parts A and B. The objective of Algorithm 3 is $\sum_{t=1}^{T} |M_t|$, where $|M_t|$ denotes the number of edges in the matching M_t. We denote the optimum solution by $\mathcal{O} = \{O_1, \ldots, O_T\}$, We have the Theorem 16 for our approximation guarantee.

Algorithm 3 Online Greedy Algorithm without Delay.

```
Input: Bipartite multigraphs on $E_1, E_2, \ldots, E_T$ on $A \cup B$ where $E_t$ is disclosed at beginning of step $t$.
Output: $\{M_1, M_2, \ldots, M_T\}$
$R_0, S \leftarrow \emptyset$, $t \leftarrow 1$.
for $t \leftarrow 1, 2, \ldots, T$ do
    $R'_t \leftarrow R_{t-1} \cup E_t$, $M_t \leftarrow argmax_{M \in \mathcal{M}, M \subseteq R'_t} |M|$.
    $S \leftarrow S \cup \{M_t\}$, $R_t \leftarrow R'_t \setminus \{M_t\}$, $t \leftarrow t + 1$.
end for
return $S$
```

▶ **Theorem 16.** *Algorithm 3 is $\frac{1}{2}$-competitive for the online circuit switch scheduling problem without delays.*

Proof. Let $\Gamma = \{E_1, \ldots, E_T\}$ denote the incoming edges for the first T steps. We call this the input sequence for the first T steps. We use induction on T to prove the theorem. Specifically, we prove that for *any* input sequence of edges for T steps, $\Gamma = \{E_1, E_2, \ldots, E_T\}$, we have $\sum_{t=1}^{T} |M_t| \geq \frac{1}{2} \sum_{t=1}^{T} |O_t|$.

For $T = 1$, we know that the maximum matching has the biggest size of any matching in the graph. So, we have $|M_1| \geq |O_1|$ and thus the base case holds. By the induction hypothesis, we have that for *any* input sequence of $T-1$ steps, we have $\sum_{t=1}^{T-1} |M_t| \geq \frac{1}{2} \sum_{t=1}^{T-1} |O_t|$ where $\{M_t\}_{t=1}^{T-1}$ and $\{O_t\}_{t=1}^{T-1}$ are the output of the algorithm and the optimal solution, respectively.

Now, consider any input sequence $E_1, \ldots, E_T$. Recall, R_1 is the residual graph formed after first step of the algorithm, i.e. $R_1 = E_1 \setminus M_1$. At the next step, the algorithm will find the maximum matching in $R'_2 = R_1 \cup E_2$ as its edge set. We build a new sequence of $T - 1$ inputs and apply induction to it.

Let $\Gamma' = \{R'_2, E_3, \ldots, E_T\}$. Consider the optimum solution on this new input sequence. Let $\{M'_t\}_{t=2}^{T}$ be the matchings that our algorithm picks given this new input sequence and $\{O'_t\}_{t=2}^{T}$ the optimum matchings. Using the induction hypothesis we can write $\sum_{t=2}^{T} |M'_t| \geq \frac{1}{2} \sum_{t=2}^{T} |O'_t|$.

First note that for $2 \leq i \leq n, M_i = M'_i$. This is true since M_i and M'_i are the maximum matchings of the same graph as can be seen inductively. We now show the following lemma that relates the optimum solution of the new instance to the original instance.

▶ **Lemma 17.** $\sum_{t=2}^{T} |O'_t| \geq \sum_{t=2}^{T} |O_t| - |M_1|$.

Proof. The matchings $\{O_2 \setminus M_1, O_3 \setminus M_1, \ldots, O_T \setminus M_1\}$ is a feasible output for the optimum solution on the Γ' sequence. Therefore, we have $\sum_{t=2}^{T} |O'_t| \geq \sum_{t=2}^{T} |O_t| - |M_1|$ as required. ◀

Using the induction hypothesis and the lemma we can write

$$\sum_{t=2}^{T} |M_t| \geq \frac{1}{2}\left(\sum_{t=2}^{T} |O_t| - |M_1|\right)$$

Adding the inequality $|M_1| \geq |O_1|$ to both sides, we obtain

$$\sum_{t=1}^{T} |M_t| \geq \frac{1}{2}\left(\sum_{t=2}^{T} |O_t|\right) + \frac{1}{2}|M_1| \geq \frac{1}{2}\left(\sum_{t=2}^{T} |O_t|\right) + \frac{1}{2}|O_1| = \frac{1}{2}\left(\sum_{t=1}^{T} |O_t|\right)$$

and the induction step follows. ◀

4.2 With Configuration Delay

In this section, we assume switching between the configurations causes a delay of $\delta \in \mathbb{N}$ steps during which no data is sent. We also assume that we have access to a β-approximation for the offline version of the problem. Note that we view the offline algorithm as a black-box. More formally, we assume we have an algorithm of the form Algorithm 4. To reiterate, G is the given complete bipartite graph, D is the traffic demand matrix, δ is the switching delay and W is the size of the time window. Recall, that sending the configuration (M, α) means that for the next α steps we will only send data using matching M.

Algorithm 4 Offline Algorithm for Circuit Switch Scheduling.

```
Input: $G = (A, B, E), D, \delta, W$
Output: $\mathcal{S} = \{(M_1, \alpha_1), \dots, (M_j, \alpha_j)\}$
```

Given a constant $k \geq 1$, the first step of the algorithm is to wait $k\delta$ steps for data to accumulate and then run the offline algorithm on the accumulated data for time window $W = k\delta$. Let $\mathcal{S}_1$ be the output of the offline algorithm. We run this schedule from time $t = k\delta + 1$ to $t = 2k\delta$. Meanwhile, we collect the incoming data matrices in these times. Figure 1 shows one step of the algorithm. At the next step, we consider the total remaining data that includes data that has not been scheduled so far from previous schedule(s) and newly arrived data in previous $k\delta$ steps. We then run the offline algorithm on this data matrix to obtain a schedule for the next $k\delta$ steps. More generally, we continue this process for every block of $k\delta$ time steps. Algorithm 5 is the formal description of the algorithm. Note that this description is written as an enumeration over blocks of size $k\delta$. Recall that $f(\mathcal{S})$ denotes the amount of data sent by any schedule $\mathcal{S}$.

Proof of Theorem 2. We use a coefficient $\gamma \leq \beta$ and optimize γ in the end. We prove the theorem by induction on the number of the blocks, i.e., l and will follow along the lines of proof of Theorem 16. As we did in the proof of Theorem 16, we consider the incoming traffic as sequences. But in this case we define a sequence $\Gamma = \{I_1, I_2, \dots, I_l\}$, where $I_i = \bigcup_{j=(i-1)(k\delta)+1}^{i(k\delta)} D_j$ is the input of block i. For $l = 1$, let the optimum schedule be $\mathcal{O}$ and the algorithm's schedule be $\mathcal{S}$. Figure 1 shows this setting. Using Lemma 10, there exists a schedule $\tilde{\mathcal{O}}$ with the property that $f\left(\tilde{\mathcal{O}}\right) \geq \left(1 - \frac{2}{k}\right) f\left(\mathcal{O}\right)$. Since $\mathcal{S}$ is the output of our offline algorithm we can write $f\left(\mathcal{S}\right) \geq \beta f\left(\mathcal{O}'\right) \geq \left(1 - \frac{2}{k}\right) \beta f\left(\mathcal{O}\right) \geq \left(1 - \frac{2}{k}\right) \gamma f\left(\mathcal{O}\right)$ and the basis of the induction is proven.

For $l = t$, again let $\mathcal{O}$ be the optimum schedule and $\mathcal{S} = S_1 \cup S_2 \cdots \cup S_t$ be the output of our algorithm where each S_i is the schedule on ith $k\delta$ block. Let O_1 be the optimum schedule for the first block and S_1 our algorithm's schedule on that block. Refer to Figure 2 for an illustration of this setting.

Algorithm 5 Online Greedy with Delay.

```
Input: $\delta, k$ and data matrices $D_1, D_2, \ldots, D_T$ on $A \times B$ where $D_i$ revealed at beginning of step $i$. Let $l = \lceil \frac{T}{k\delta} \rceil$.
Output: $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_l$.
$\mathcal{S} \leftarrow \emptyset$, $R_0 \leftarrow \emptyset$.
for $r \leftarrow 0, \ldots, l-1$ do
    $R'_r \leftarrow R_r + \sum_{rk\delta+1 \leq j \leq (r+1)k\delta} D_j$.
    $\mathcal{S}_r \leftarrow OfflineAlgorithm\,(G, R'_r, \delta, k\delta)$.
    $R_{r+1} \leftarrow R'_r - min\left(R'_r, \sum_{(\alpha,M)\in\mathcal{S}_r} \alpha M\right)$, $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_r$.
end for
return $\mathcal{S}$
```

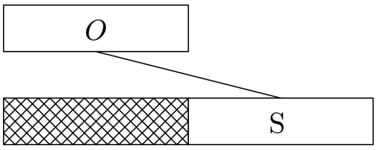


Figure 1 Basis of the induction. The crossed out block is the waiting period of our algorithm.

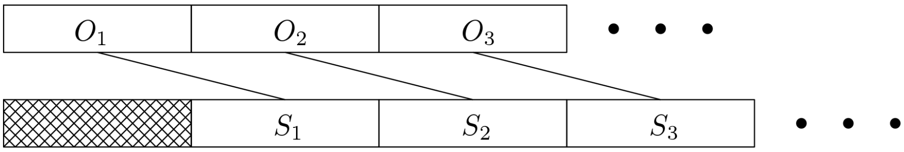


Figure 2 Step of the induction.

Consider the new input sequence $\Gamma' = \{R'_1 \cup I_2, I_3, \ldots, I_t\}$. Let the optimum schedule on the new input sequence be $\mathcal{O}'$ and the algorithm's schedule be $\mathcal{S}' = S'_1 \cup \cdots \cup S'_l$. From the induction hypothesis, we have $f(\mathcal{S}') \geq \left(1 - \frac{2}{k}\right) \gamma f(\mathcal{O}')$. Note that $S_i = S'_{i-1}$ for $i \geq 2$ and thus $f(\mathcal{S}') = f(\mathcal{S} \setminus S_1) = f(\mathcal{S}) - f(S_1)$. As in the proof of Lemma 17, a candidate schedule for the new instance is to consider $\mathcal{O} \setminus O_1$ and ignore the data sent by the algorithm in the schedule S_1 if it appears in any of the optimal matchings. Thus we obtain that

$$f(\mathcal{O}') \geq f(\mathcal{O} \setminus O_1) - f(S_1) = f(\mathcal{O}) - f(O_1) - f(S_1).$$

For O_1 based on our basis argument we can find S_1 such that $f(S_1) \geq \left(1 - \frac{2}{k}\right) \beta f(O_1)$. To sum up, we have the two following inequalities:

$$\begin{aligned} f(\mathcal{S}) - f(S_1) &\geq \left(1 - \frac{2}{k}\right) \gamma \left((f(\mathcal{O}) - f(O_1)) - f(S_1) \right), \\ f(S_1) &\geq \left(1 - \frac{2}{k}\right) \beta f(O_1). \end{aligned}$$

Rewriting the first inequality, we have

$$f(\mathcal{S}) - \left(1 - \left(1 - \frac{2}{k}\right)\gamma\right) f(S_1) \geq \left(1 - \frac{2}{k}\right) \gamma \left(f(\mathcal{O}) - f(O_1)\right)$$

Adding the $\left(1 - \left(1 - \frac{2}{k}\right)\gamma\right)$ times the second inequality

$$f(\mathcal{S}) \geq \left(1 - \frac{2}{k}\right) \gamma f(\mathcal{O}) - \left(1 - \frac{2}{k}\right) \left(\gamma - \beta \left(1 - \left(1 - \frac{2}{k}\right)\gamma\right)\right) f(O_1)$$

Optimizing the γ we get $\gamma = \frac{\beta}{(1+(1-\frac{2}{k})\beta)}$ and thus proving the theorem. ◀

References

1 Nir Andelman and Yishay Mansour. Auctions with budget constraints. In *Scandinavian Workshop on Algorithm Theory*, pages 26–38, 2004.

2 Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1497–1514, 2014.

3 Siddharth Barman. Approximating Nash Equilibria and Dense Bipartite Subgraphs via an Approximate Version of Caratheodory's Theorem. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 361–369, 2015.

4 G Celik, Sem C Borst, Philip A Whiting, and Eytan Modiano. Dynamic scheduling with reconfiguration delays. *Queueing Systems*, 83(1-2):87–129, 2016.

5 Cheng-Shang Chang, Wen-Jyh Chen, and Hsiang-Yi Huang. On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by Birkhoff and von Neumann. In *1999 Seventh International Workshop on Quality of Service (IWQoS'99)*, pages 79–86, 1999.

6 K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. OSA: An Optical Switching Architecture for Data Center Networks With Unprecedented Flexibility. *IEEE/ACM Transactions on Networking*, 22(2):498–511, 2014.

7 Abel Dasylva and R Srikant. Optimal WDM schedules for optical star networks. *IEEE/ACM Transactions on Networking*, 7(3):446–456, 1999.

8 Fanny Dufossé, Kamer Kaya, Ioannis Panagiotas, and Bora Uçar. Further notes on Birkhoff–von Neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 554:68–78, 2018.

9 Alina Ene and Huy L. Nguyen. A Nearly-linear Time Algorithm for Submodular Maximization with a Knapsack Constraint. *CoRR*, abs/1709.09767, 2017. arXiv:1709.09767.

10 Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):339–350, 2010.

11 Shu Fu, Bin Wu, Xiaohong Jiang, Achille Pattavina, Lei Zhang, and Shizhong Xu. Cost and delay tradeoff in three-stage switch architecture for data center networks. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pages 56–61, 2013.

12 Leonidas Georgiadis, Michael J Neely, Leandros Tassiulas, et al. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends® in Networking*, 1(1):1–144, 2006.

13 Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R Das, Jon P Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *ACM SIGCOMM Computer Communication Review*, volume 44 (4), pages 319–330, 2014.

14 Thomas Inukai. An efficient SS/TDMA time slot assignment algorithm. *IEEE Transactions on Communications*, 27(10):1449–1455, 1979.

15 Srikanth Kandula, Jitendra Padhye, and Victor Bahl. Flyways to DeCongest Data Center Networks. *Proc. of Hot Nets*, 2009.

16 Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.

17 Janardhan Kulkarni, Euiwoong Lee, and Mohit Singh. Minimum Birkhoff-von Neumann Decomposition. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 343–354, 2017.

18 Conglong Li, Matthew K. Mukerjee, David G. Andersen, Srinivasan Seshan, Michael Kaminsky, George Porter, and Alex C. Snoeren. Using Indirect Routing to Recover from Network Traffic Scheduling Estimation Error. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, ANCS '17, pages 13–24, 2017.

19 Xin Li and Mounir Hamdi. On scheduling optical packet switches with reconfiguration delay. *IEEE Journal on Selected Areas in Communications*, 21(7):1156–1164, 2003.

20 He Liu, Matthew K Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M Voelker, David G Andersen, Michael Kaminsky, et al. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, page 41, 2015.

21 Liang Liu, Long Gong, Sen Yang, Jun Xu, and Lance Fortnow. Better Algorithms for Hybrid Circuit and Packet Switching in Data Centers. *arXiv preprint*, 2017. `arXiv:1712.06634`.

22 Vahab Mirrokni, Renato Paes Leme, Adrian Vladu, and Sam Chiu wai Wong. Tight Bounds for Approximate Carathéodory and Beyond. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2440–2448, 2017.

23 Roy Schwartz, Mohit Singh, and Sina Yazdanbod. Online and Offline Greedy Algorithms for Routing with Switching Costs. *arXiv preprint*, 2019. `arXiv:1905.02800`.

24 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

25 Brian Towles and William J Dally. Guaranteed scheduling for switches with configuration overhead. *IEEE/ACM Transactions on Networking*, 11(5):835–847, 2003.

26 Shay Vargaftik, Katherine Barabash, Yaniv Ben-Itzhak, Ofer Biran, Isaac Keslassy, Dean Lorenz, and Ariel Orda. Composite-path switching. In *Proceedings of the 12th International on Conference on emerging Networking Experiments and Technologies*, pages 329–343, 2016.

27 S. Bojja Venkatakrishnan, M. Alizadeh, and P. Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. *Queueing Systems*, pages 1–37, 2018.

28 Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Ng, Michael Kozuch, and Michael Ryan. c-Through: Part-time optics in data centers. In *ACM SIGCOMM Computer Communication Review*, volume 40 (4), pages 327–338, 2010.

29 Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y Zhao, and Haitao Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):443–454, 2012.

On the Probabilistic Degrees of Symmetric Boolean Functions

Srikanth Srinivasan
Department of Mathematics, Indian Institute of Technology Bombay, Mumbai, India
srikanth@math.iitb.ac.in

Utkarsh Tripathi
Department of Mathematics, Indian Institute of Technology Bombay, Mumbai, India
utkarshtripathi.math@gmail.com

S. Venkitesh
Department of Mathematics, Indian Institute of Technology Bombay, Mumbai, India
venkitesh.mail@gmail.com

Abstract

The probabilistic degree of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is defined to be the smallest d such that there is a random polynomial $\mathbf{P}$ of degree at most d that agrees with f at each point with high probability. Introduced by Razborov (1987), upper and lower bounds on probabilistic degrees of Boolean functions – specifically symmetric Boolean functions – have been used to prove explicit lower bounds, design pseudorandom generators, and devise algorithms for combinatorial problems.

In this paper, we characterize the probabilistic degrees of all symmetric Boolean functions up to polylogarithmic factors over all fields of fixed characteristic (positive or zero).

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Symmetric Boolean function, probabilistic degree

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.28

Related Version A full version of the paper is available on [21], `https://eccc.weizmann.ac.il/report/2019/138/download`.

Funding *Srikanth Srinivasan*: Supported by MATRICS grant MTR/2017/000958 awarded by SERB, Government of India.
Utkarsh Tripathi: Supported by the Ph.D. Scholarship of NBHM, DAE, Government of India.
S. Venkitesh: Supported by the Senior Research Fellowship of HRDG, CSIR, Government of India.

1 Introduction

Studying the combinatorial and computational properties of Boolean functions by representing them using multivariate polynomials (over some field $\mathbb{F}$) is an oft-used technique in Theoretical Computer Science. Such investigations into the complexity of Boolean functions have led to many important advances in the area (see, e.g. [2, 14, 23] for a large list of such results).

An "obvious" way of representing a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is via a multilinear polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$ such that $P(a) = f(a)$ for all $a \in \{0,1\}^n$. While such a representation has the advantage of being unique, understanding the computational complexity of f sometimes requires us to understand polynomial representations where we allow some notion of error in the representation. Here again, many kinds of representations have been studied, but we concentrate here on the notion of *Probabilistic degree* of a Boolean function, introduced by Razborov [16]. It is defined as follows.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 28; pp. 28:1–28:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Definition 1** (Probabilistic polynomial and Probabilistic degree). *Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and an $\varepsilon > 0$, an ε-*error probabilistic polynomial *for f is a random polynomial $\mathbf{P}$ (with some distribution having finite support) over $\mathbb{F}[x_1, \ldots, x_n]$ such that for each $a \in \{0,1\}^n$,*

$$\Pr_{\mathbf{P}}[\mathbf{P}(a) \neq f(a)] \leq \varepsilon.$$

We say that the degree of $\mathbf{P}$, denoted $\deg(\mathbf{P})$*, is at most d if the probability distribution defining $\mathbf{P}$ is supported on polynomials of degree at most d. Finally, we define the ε-*error probabilistic degree *of f, denoted* $\mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(f)$*, to be the least d such that f has an ε-error probabilistic polynomial of degree at most d.*

When the field $\mathbb{F}$ is clear from context, we use $\mathrm{pdeg}_{\varepsilon}(f)$ *instead of* $\mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(f)$.

Intuitively, if we think of multivariate polynomials as algorithms and degree as a notion of efficiency, then a low-degree probabilistic polynomial for a Boolean function f is an efficient *randomized* algorithm for f.

The study of the probabilistic degree itself is by now a classical topic, and has had important repercussions for other problems. We list three such examples below, referring the reader to the papers for definitions and exact statements of the results.

- Razborov [16] showed strong upper bounds on the probabilistic degree of the OR function over fields of (fixed) positive characteristic. Along with lower bounds on the probabilistic degree of some symmetric Boolean functions,[1] this led to the first lower bounds for the Boolean circuit class $\mathrm{AC}^0[p]$, for prime p [16, 17, 19].
- Tarui [22] and Beigel, Reingold and Spielman [3] showed upper bounds on the probabilistic degree of the OR function over any characteristic (and in particular over the reals). This leads to probabilistic degree upper bounds for the circuit class AC^0, which was used by Braverman [5] to resolve a long-standing open problem of Linial and Nisan [11] regarding pseudorandom generators for AC^0.
- Alman and Williams [1] showed that for constant error, the probabilistic degree of any symmetric Boolean function is at most $O(\sqrt{n})$, and used this to obtain the first subquadratic algorithm for an offline version of the Nearest Neighbour problem in the Hamming metric.

In all the above results, it was important to understand the probabilistic degree of a certain class of symmetric Boolean functions. However, the problem of *characterizing* the probabilistic degree of symmetric Boolean functions in general does not seem to have been considered. This is somewhat surprising, since this problem has been considered in a variety of other computational models, such as AC^0 circuits of polynomial size [8, 6], $\mathrm{AC}^0[p]$ circuits of quasipolynomial size [12], Approximate degree[2] [15] and Perceptrons[3] of quasipolynomial size [24].

[1] Recall that a *symmetric* Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a function such that $f(x)$ depends only on the Hamming weight of x. Examples include the threshold functions, Parity (counting modulo 2), etc.

[2] A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is said to have approximate degree at most d if there is a degree d polynomial $P \in \mathbb{R}[x_1, \ldots, x_n]$ such that at each $a \in \{0,1\}^n$, $|f(a) - P(a)| \leq 1/4$.

[3] Perceptrons are depth-2 circuits with a Majority gate as the output gate with AND and OR gates feeding into it.

Our result. In this paper, we give an almost-complete understanding of the probabilistic degrees of all symmetric Boolean functions over all fields of fixed positive characteristic and characteristic 0. For each Boolean function f on n variables, our upper bounds and lower bounds on $\mathrm{pdeg}(f)$ are separated only by polylogarithmic factors in n.

We now introduce some notation and give a formal statement of our result. We shall use the notation $[a, b]$ to denote an interval in $\mathbb{R}$ as well as an interval in $\mathbb{Z}$; the distinction will be clear from the context. Throughout, fix some field $\mathbb{F}$ of characteristic p which is either a fixed positive constant or 0. Let n be a growing integer parameter which will always be the number of input variables. We use $s\mathcal{B}_n$ to denote the set of all symmetric Boolean functions on n variables. Note that each symmetric Boolean function $f : \{0,1\}^n \to \{0,1\}$ is uniquely specified by a string $\mathrm{Spec}\, f : [0,n] \to \{0,1\}$, which we call the *Spectrum* of f, in the sense that for any $a \in \{0,1\}^n$, we have

$$f(a) = \mathrm{Spec}\, f(|a|).$$

Given a $f \in s\mathcal{B}_n$, we define the *period of* f, denoted $\mathrm{per}(f)$, to be the smallest positive integer b such that $\mathrm{Spec}\, f(i) = \mathrm{Spec}\, f(i+b)$ for all $i \in [0, n-b]$. We say f is k*-bounded* if $\mathrm{Spec}\, f$ is constant on the interval $[k, n-k]$; let $B(f)$ denote the smallest k such that f is k-bounded.

Standard decomposition of a symmetric Boolean function [12]. Fix any $f \in s\mathcal{B}_n$. Among all symmetric Boolean functions $f' \in s\mathcal{B}_n$ such that $\mathrm{Spec}\, f'(i) = \mathrm{Spec}\, f(i)$ for all $i \in [\lceil n/3 \rceil, \lfloor 2n/3 \rfloor]$, we choose a function g such that $\mathrm{per}(g)$ is as small as possible. We call g the *periodic part* of f. Define $h \in s\mathcal{B}_n$ by $h = f \oplus g$. We call h the *bounded part* of f.

We will refer to the pair (g, h) as a *standard decomposition* of the function f. Note that we have $f = g \oplus h$.

▶ **Observation 2.** *Let $f \in s\mathcal{B}_n$ and let (g, h) be a standard decomposition of f. Then,* $\mathrm{per}(g) \le \lfloor n/3 \rfloor$ *and* $B(h) \le \lceil n/3 \rceil$.

In this paper, we prove the following upper and lower bounds for the probabilistic degrees of symmetric Boolean functions. While the most important setting for understanding the probabilistic degree is the setting of constant error (i.e. $\varepsilon = \Omega(1)$), we state the upper bound results for arbitrary $\varepsilon > 0$ since the inductive construction naturally gives rise to this stronger statement.

▶ **Theorem 3** (Upper bounds on probabilistic degree)**.** *Let $\mathbb{F}$ be a field of constant characteristic p (possibly 0) and $n \in \mathbb{N}$ be a growing parameter. Let $f \in s\mathcal{B}_n$ be arbitrary and let (g, h) be a standard decomposition of f. Then we have the following for any $\varepsilon > 0$.*

1. *If* $\mathrm{per}(g) = 1$, *then* $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(g) = 0,$.
   If $\mathrm{per}(g)$ *is a power of p, then* $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(g) \le \mathrm{per}(g)$, *[12]*
   (Note that $\mathrm{per}(g)$ *cannot be a power of p if $p = 0$.)*
2. $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(h) = \widetilde{O}(\sqrt{B(h)\log(1/\varepsilon)} + \log(1/\varepsilon))$,
3. $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(f) = \begin{cases} O(\sqrt{n\log(1/\varepsilon)}) & \textit{if } \mathrm{per}(g) > 1 \textit{ and not a power of } p\textit{, [1]} \\ \widetilde{O}(\min\{\sqrt{n\log(1/\varepsilon)}, \mathrm{per}(g) + & \textit{otherwise.} \\ \quad \sqrt{B(h)\log(1/\varepsilon)} + \log(1/\varepsilon)\}) & \end{cases}$

When p is positive, we can replaced the $\widetilde{O}(\cdot)$ with $O(\cdot)$ in all the above bounds.

We obtain almost (up to polylogarithmic factors) matching lower bounds for all symmetric Boolean functions over all fields.

▶ **Theorem 4** (Lower bounds on probabilistic degree). *Let $\mathbb{F}$ be a field of constant characteristic p (possibly 0) and $n \in \mathbb{N}$ be a growing parameter. Let $f \in s\mathcal{B}_n$ be arbitrary and let (g, h) be a standard decomposition of f. Then for any* constant $\varepsilon \leq 1/3$*, we have*

1. $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(g) = \widetilde{\Omega}(\sqrt{n})$ *if* $\mathrm{per}(g) > 1$ *and is not a power of* p *and* $\widetilde{\Omega}(\min\{\sqrt{n}, \mathrm{per}(g)\})$ *otherwise.*
2. $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(h) = \widetilde{\Omega}(\sqrt{B(h)})$,
3. $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(f) = \begin{cases} \widetilde{\Omega}(\sqrt{n}) & \textit{if } \mathrm{per}(g) > 1 \textit{ and not a power of } p, \\ \widetilde{\Omega}(\min\{\sqrt{n}, \mathrm{per}(g) + \sqrt{B(h)}\}) & \textit{otherwise.} \end{cases}$

where the $\widetilde{\Omega}(\cdot)$ *hides* $\mathrm{poly}(\log n)$ *factors.*

▶ Remark 5. A natural open question following our results is to remove the polylogarithmic factors separating our upper and lower bounds. We remark that in characteristic 0, such gaps exist even for the very simple OR function despite much effort [13, 9, 4]. Over positive characteristic, there is no obvious barrier, but our techniques fall short of proving tight lower bounds for natural families of functions such as the Exact Threshold functions (defined below).

Many proofs are omitted for lack of space. They appear in the full version of the paper.

1.1 Proof Outline

For the outline below, we assume that the field is of fixed positive characteristic p.

Upper bounds. Given a symmetric Boolean function f on n variables with standard decomposition (g, h), it is easy to check that $\mathrm{pdeg}_\varepsilon(f) = O(\mathrm{pdeg}_\varepsilon(g) + \mathrm{pdeg}_\varepsilon(h))$. So it suffices to upper bound the probabilistic degrees of periodic and bounded functions respectively.

For periodic functions g with period a power of p, Lu [12] showed that the *exact* degree of the Boolean functions is at most $\mathrm{per}(g)$. If the period is not a power of p, then we use the upper bound of Alman and Williams [1] that holds for all symmetric Boolean functions (as we show below, this is nearly the best that is possible).

For a t-constant function h (defined in Section 3), we use the observation that any t-constant function is essentially a linear combination of the threshold functions $\mathrm{Thr}_n^0, \ldots, \mathrm{Thr}_n^t$ (see Section 2 for the definition) and so it suffices to construct probabilistic polynomials for Thr_n^i, for $i \in [0, t]$.[4]

Our main technical upper bound is a new probabilistic degree upper bound of $O(\sqrt{t \log(1/\varepsilon)} + \log(1/\varepsilon))$ for any threshold function Thr_n^t. This upper bound interpolates smoothly between a classical upper bound of $O(\log(1/\varepsilon))$ due to Razborov [16] for $t = 1$ and a recent result of Alman and Williams [1] that yields $O(\sqrt{n \log(1/\varepsilon)})$ for $t = \Omega(n)$.

The proof of our upper bound is based on the beautiful inductive construction of Alman and Williams [1] which gives their above-mentioned result. The key difference between our proof and the proof of [1] is that we need to handle separately the case when the error $\varepsilon \leq 2^{-\Omega(t)}$.[5] In [1], this is a trivial case since any function on n Boolean variables has an exact polynomial of degree n which is at most $O(\sqrt{n \log(1/\varepsilon)})$ when $\varepsilon \leq 2^{-\Omega(n)}$. In our setting, the correct bound in this case is $O(\log(1/\varepsilon))$, which is non-obvious. We obtain this bound by a suitable modification of Razborov's technique (for $t = 1$) to handle larger thresholds.

[4] We actually need to construct probabilistic polynomials for all the threshold functions simultaneously. We ignore this point in this high-level outline.

[5] This case comes up naturally in the inductive construction, even if one is ultimately only interested in the case when ε is a constant.

Lower bounds. Here, our proof closely follows a result of Lu [12], who gave a characterization of symmetric Boolean functions that have quasipolynomial-sized $\mathrm{AC}^0[p]$ circuits.[6] To show circuit lower bounds for a symmetric Boolean function h, Lu showed how to convert a circuit C computing h to a circuit C' computing either the Majority or a MOD_q function (where q and p are relatively prime). Since both of these are known to be hard for $\mathrm{AC}^0[p]$ [16, 17], we get the lower bound.

We show how to use Lu's reductions (and variants thereof) but in the setting of probabilistic polynomials. This works because

- We also have strong probabilistic degree lower bounds for the Majority and MOD_q functions (in fact, this is the source of the $\mathrm{AC}^0[p]$ lower bound).
- Lu's constructions of the hard functions from h (and our variants) involve taking ANDs and ORs of a few copies of (restrictions of) h. This also gives a reduction from the hard functions to h in the setting of probabilistic degree, since ANDs and ORs are known to have small probabilistic degree [16].

With these observations in place, the proof reduces to a careful case analysis to get the correct lower bound in each case. Interestingly, while it is not clear whether these ideas give a tight understanding of the $\mathrm{AC}^0[p]$-circuit complexity of symmetric Boolean functions, they do give nearly tight (up to log factors) lower bounds for probabilistic degree.

2 Preliminaries

Some Boolean functions. Fix some positive $n \in \mathbb{N}$. The *Majority* function Maj_n on n Boolean variables accepts exactly the inputs of Hamming weight at least $n/2$. For $t \in [0, n]$, the *Threshold* function Thr_n^t accepts exactly the inputs of Hamming weight at least t; and similarly, the *Exact Threshold* function EThr_n^t accepts exactly the inputs of Hamming weight exactly t. Finally, for $b \in [2, n]$ and $i \in [0, b-1]$, the function $\mathrm{MOD}_n^{b,i}$ accepts exactly those inputs a such that $|a| \equiv i \pmod{b}$. In the special case that $i = 0$, we also use MOD_n^b.

▶ **Fact 6.** *We have the following simple facts about probabilistic degrees. Let $\mathbb{F}$ be any field.*

1. *(Error reduction [9]) For any $\delta < \varepsilon \le 1/3$ and any Boolean function f, if $\mathbf{P}$ is an ε-error probabilistic polynomial for f, then $\mathbf{Q} = M(\mathbf{P}_1, \ldots, \mathbf{P}_\ell)$ is a δ-error probabilistic polynomial for f where M is the exact multilinear polynomial for Maj_ℓ and $\mathbf{P}_1, \ldots, \mathbf{P}_\ell$ are independent copies of $\mathbf{P}$. In particular, we have $\mathrm{pdeg}_\delta^{\mathbb{F}}(f) \le \mathrm{pdeg}_\varepsilon^{\mathbb{F}}(f) \cdot O(\log(1/\delta)/\log(1/\varepsilon))$.*
2. *(Composition) For any Boolean function f on k variables and any Boolean functions $g_1, \ldots, g_k$ on a common set of m variables, let h denote the natural composed function $f(g_1, \ldots, g_k)$ on m variables. Then, for any $\varepsilon, \delta > 0$, we have $\mathrm{pdeg}_{\varepsilon+k\delta}^{\mathbb{F}}(h) \le \mathrm{pdeg}_\varepsilon^{\mathbb{F}}(f) \cdot \max_{i\in[k]} \mathrm{pdeg}_\delta^{\mathbb{F}}(g_i)$.*
3. *(Sum) Assume that $f, g_1, \ldots, g_k$ are all Boolean functions on a common set of m variables such that $f = \sum_{i\in[k]} g_i$. Then, for any $\delta > 0$, we have $\mathrm{pdeg}_{k\delta}^{\mathbb{F}}(f) \le \max_{i\in[k]} \mathrm{pdeg}_\delta^{\mathbb{F}}(g_i)$.*

2.1 Some previous results on probabilistic degree

The following upper bounds on probabilistic degrees of OR and AND functions were proved by Razborov [16] in the case of positive characteristic and Tarui [22] and Beigel, Reingold and Spielman [3] in the general case.

[6] Recall that an $\mathrm{AC}^0[p]$ circuit is a constant-depth circuit made up of gates that can compute the Boolean functions AND, OR, NOT and MOD_p (defined below).

▶ **Lemma 7** (Razborov's upper bound on probabilistic degrees of OR and AND)**.** *Let $\mathbb{F}$ be a field of characteristic p. For $p > 0$, we have*

$$\mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(\mathrm{OR}_n) = \mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(\mathrm{AND}_n) = O(p\log(1/\varepsilon)). \tag{1}$$

For any p, we have

$$\mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(\mathrm{OR}_n) = \mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(\mathrm{AND}_n) = O(\log n \cdot \log(1/\varepsilon)). \tag{2}$$

We now recall two probabilistic degree lower bounds due to Smolensky [18, 20], building on the work of Razborov [16].

▶ **Lemma 8** (Smolensky's lower bound for close-to-Majority functions)**.** *For any field $\mathbb{F}$, any $\varepsilon \in (1/2^n, 1/5)$, and any Boolean function g on n variables that agrees with* Maj_n *on a $1-\varepsilon$ fraction of its inputs, we have*

$$\mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(g) = \Omega(\sqrt{n\log(1/\varepsilon)}).$$

▶ **Lemma 9** (Smolensky's lower bound for MOD functions)**.** *For $2 \le b \le n/2$, any $\mathbb{F}$ such that char$(\mathbb{F}) = p$ is coprime to b, any $\varepsilon \in (1/2^n, 1/(3b))$, there exists an $i \in [0, b-1]$ such that*

$$\mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(\mathrm{MOD}_n^{b,i}) = \Omega(\sqrt{n\log(1/b\varepsilon)}).$$

▶ Remark 10. From the above lemma, it also easily follows that if $b \le n/4$, then for *every* $i \in [0, b-1]$, we have $\mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(\mathrm{MOD}_n^{b,i}) = \Omega(\sqrt{n\log(1/b\varepsilon)})$. This is the usual form in which Smolensky's lower bound is stated. The above form is slightly more useful to us because it holds for b up to $n/2$.

We will also need the following result of Alman and Williams [1].

▶ **Lemma 11.** *Let $\mathbb{F}$ be any field. For any $n \ge 1, \varepsilon > 0$ and $f \in s\mathcal{B}_n$,* $\mathrm{pdeg}_{\varepsilon}^{\mathbb{F}}(f) = O(\sqrt{n\log(1/\varepsilon)})$.

2.2 A string lemma

Given a function $w : I \to \{0,1\}$ where $I \subseteq \mathbb{N}$ is an interval, we think of w as a string from the set $\{0,1\}^{|I|}$ in the natural way. For an interval $J \subseteq I$, we denote by $w|_J$ the substring of w obtained by restriction to J.

The following simple lemma can be found, e.g. as a consequence of [10, Chapter I, Section 2, Theorem 1].

▶ **Lemma 12.** *Let $w \in \{0,1\}^+$ be any non-empty string and $u, v \in \{0,1\}^+$ such that $w = uv = vu$. Then there exists a string $z \in \{0,1\}^+$ such that w is a power of z (i.e. $w = z^k$ for some $k \ge 2$).*

▶ **Corollary 13.** *Let $g \in s\mathcal{B}_n$ be arbitrary with* $\mathrm{per}(g) = b > 1$. *Then for all $i, j \in [0, n-b+1]$ such that $i \not\equiv j \pmod b$, we have* $\mathrm{Spec}\, g|_{[i,i+b-1]} \neq \mathrm{Spec}\, g|_{[j,j+b-1]}$.

Proof. Suppose $\mathrm{Spec}\, g|_{[i,i+b-1]} = \mathrm{Spec}\, g|_{[j,j+b-1]}$ for some $i \not\equiv j \pmod b$. Assume without loss of generality that $i < j < i+b$. Let $u = \mathrm{Spec}\, g|_{[i,j-1]}, v = \mathrm{Spec}\, g|_{[j,i+b-1]}, w = \mathrm{Spec}\, g|_{[i+b,j+b-1]}$. Then $u = w$ and the assumption $uv = vw$ implies $uv = vu$. By Lemma 12, there exists a string z such that $uv = z^k$ for $k \ge 2$ and therefore $\mathrm{per}(g) < b$. This contradicts our assumption on b. ◀

▶ **Lemma 14.** *Let $n \in \mathbb{N}$ be a growing parameter and let $f \in s\mathcal{B}_n$ with periodic part g. For any $1 \leq b \leq \lfloor n/3 \rfloor$, either $\mathrm{per}(g) \leq b$ or for all distinct $i, j \in [\lceil n/3 \rceil - b, \lceil n/3 \rceil]$, $\mathrm{Spec}\, f|_{[i,i+(\lceil n/3 \rceil + b)]} \neq \mathrm{Spec}\, f|_{[j,j+(\lceil n/3 \rceil + b)]}$.*

Proof. W.l.o.g. say $i < j$. Assume $\mathrm{per}(g) > b$ (otherwise, we are done trivially). Then, for any $b' \leq b$, it follows that there is an $k \in [\lceil n/3 \rceil, \lfloor 2n/3 \rfloor - b']$ such that $\mathrm{Spec}\, f(k) \neq \mathrm{Spec}\, f(k + b')$. In particular, we see that $\mathrm{Spec}\, f|_{[i,i+(\lceil n/3 \rceil + b)]} \neq \mathrm{Spec}\, f|_{[i+b',i+b'+(\lceil n/3 \rceil + b)]}$. Fixing $b' = j - i$ yields the result. ◀

3 Upper bounds

In this section, we will first prove upper bounds on the probabilistic degree of a smaller class of symmetric Boolean functions, called *t-constant functions*, and then use it to prove Theorem 3.

3.1 Upper bound on probabilistic degree of t-constant functions

▶ **Definition 15** (t-constant function)**.** *For any positive $n \in \mathbb{N}$ and $t \in [0, n]$, a Boolean function $f \in s\mathcal{B}_n$ is said to be* t-constant *if $f|_{\{x:|x| \geq t\}}$ is a constant, that is, $\mathrm{Spec}\, f|_{[t,n]}$ is a constant.*

The following observation is immediate.

▶ **Observation 16.** *A Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ is t-constant if and only if $f = \sum_{j=0}^{t} a_j \mathrm{Thr}_n^j$, for some $a_0, \ldots, a_t \in \{-1, 0, 1\}$. In other words, f is t-constant if and only if there exists a linear polynomial $g(Y_0, \ldots, Y_t) = a_0 Y_0 + \cdots + a_t Y_t \in \mathbb{F}[Y_0, \ldots, Y_t]$ with $a_j \in \{-1, 0, 1\}$, $j \in [0, t]$ such that $f = g(\mathrm{Thr}_n^0, \ldots, \mathrm{Thr}_n^t)$.*

We will prove an upper bound on the probabilistic degree of t-constant Boolean functions. For this, we first generalize the notion of probabilistic polynomial and probabilistic degree to a *tuple* of Boolean functions. This generalization was implicit in [1].

▶ **Definition 17** (Probabilistic poly-tuple and probabilistic degree)**.** *Let $f = (f_1, \ldots, f_m) : \{0, 1\}^n \to \{0, 1\}^m$ be an m-tuple of Boolean functions and $\varepsilon \in (0, 1)$. An ε-error probabilistic poly-tuple for f is a random m-tuple of polynomials $\mathbf{P}$ (with some distribution having finite support) from $\mathbb{F}[X_1, \ldots, X_n]^m$ such that*

$$\Pr_{P \sim \mathbf{P}} [P(x) \neq f(x)] \leq \varepsilon, \quad \textit{for all } x \in \{0, 1\}^n.$$

We say that the degree of $\mathbf{P}$ is at most d if $\mathbf{P}$ is supported on m-tuples of polynomials $P = (P_1, \ldots, P_m)$ where each P_i has degree at most d. Finally we define the ε-error probabilistic degree of f, denoted by $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(f)$, to be the least d such that f has an ε-error probabilistic poly-tuple of degree at most d.

We make a definition for convenience.

▶ **Definition 18** (Threshold tuple)**.** *For positive $n \in \mathbb{N}$, $t \in [0, n]$, an (n, t)-threshold tuple is any tuple of Boolean functions $(\mathrm{Thr}_n^{t_1}, \ldots, \mathrm{Thr}_n^{t_m})$, with $t_1, \ldots, t_m \in [0, t]$ and $\max\{t_1, \ldots, t_m\} \leq t$.*

The main theorem of this subsection is the following.

▶ **Theorem 19.** *For any positive $n \in \mathbb{N}$, $t \in [0,n]$, if T is an (n,t)-threshold tuple and $\varepsilon \in (0,1/3)$, then*

$$\mathrm{pdeg}_\varepsilon(T) = \begin{cases} \widetilde{O}(\sqrt{t\log(1/\varepsilon)} + \log(1/\varepsilon)), & \mathit{char}(\mathbb{F}) = 0, \\ O(\sqrt{t\log(1/\varepsilon)} + \log(1/\varepsilon)), & \mathit{char}(\mathbb{F}) = p > 0. \end{cases}.$$

As a corollary to the above theorem, we get an upper bound for the probabilistic degree of t-constant functions.

▶ **Corollary 20.** *For any t-constant Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $\varepsilon \in (0,1/3)$,*

$$\mathrm{pdeg}_\varepsilon(f) = \begin{cases} \widetilde{O}(\sqrt{t\log(1/\varepsilon)} + \log(1/\varepsilon)), & \mathit{char}(\mathbb{F}) = 0, \\ O(\sqrt{t\log(1/\varepsilon)} + \log(1/\varepsilon)), & \mathit{char}(\mathbb{F}) = p > 0. \end{cases}.$$

Proof. By Observation 16, there exists $g(Y_0,\ldots,Y_t) = a_0Y_0 + \cdots + a_tY_t \in \mathbb{F}[Y_0,\ldots,Y_t]$ with $a_j \in \{-1,0,1\}$, $j \in [0,t]$ such that $f = g(\mathrm{Thr}_n^0,\ldots,\mathrm{Thr}_n^t)$. We note that $\deg g = 1$. So by Theorem 19, we get

$$\mathrm{pdeg}_\varepsilon(f) = \deg g \cdot \mathrm{pdeg}_\varepsilon(\mathrm{Thr}_n^0,\ldots,\mathrm{Thr}_n^t) = \begin{cases} \widetilde{O}(\sqrt{t\log(1/\varepsilon)} + \log(1/\varepsilon)), & \mathrm{char}(\mathbb{F}) = 0, \\ O(\sqrt{t\log(1/\varepsilon)} + \log(1/\varepsilon)), & \mathrm{char}(\mathbb{F}) = p > 0. \end{cases}.$$

◀

Before we prove Theorem 19, we will gather a few results that we require. The following lemma is a particular case of Bernstein's inequality (Theorem 1.4, [7]).

▶ **Lemma 21.** *Let $X_1,\ldots,X_m$ be independent and identically distributed Bernoulli random variables with mean p. Let $X = \sum_{i=1}^m X_i$. Then for any $\theta > 0$,*

$$\Pr\left[|X - mp| > \theta\right] \le 2\exp\left(-\frac{\theta^2}{2mp(1-p) + 2\theta/3}\right).$$

We will also need the following polynomial construction.

▶ **Theorem 22** (Lemma 3.1, [1])**.** *For any symmetric Boolean function $f : \{0,1\}^n \to \{0,1\}$ and integer interval $[a,b] \subseteq [0,n]$, there exists a symmetric multilinear polynomial $EX_{[a,b]}f \in \mathbb{Z}[X_1,\ldots,X_n]$ such that $\deg(EX_{[a,b]}f) \le b-a$ and $\mathrm{Spec}\,(EX_{[a,b]}f)|_{[a,b]} = \mathrm{Spec}\,f|_{[a,b]}$.*

We will now prove Theorem 19.

Proof of Theorem 19. For any $a = (a_1,\ldots,a_k), b = (b_1,\ldots,b_k) \in \mathbb{F}^k$, fix the notation $a * b = (a_1b_1,\ldots,a_kb_k)$. Throughout, the notation $\mathbf{1}$ will denote the constant-1 vector of appropriate length.

For positive characteristic p, we prove that for any positive $n \in \mathbb{N}$, $t \in [0,n]$ and $\varepsilon \in (0,1/100)$, any (n,t)-threshold tuple T has an ε-error probabilistic poly-tuple $\mathbf{T}$ of degree at most $A_p\sqrt{t\log(1/\varepsilon)} + B_p\log(1/\varepsilon)$, for constants $A_p = B_p = 4800000p$ (we make no effort to minimize the constants). For $p = 0$, we prove a similar result with a degree bound of $A_0 \log n \cdot \sqrt{t\log(1/\varepsilon)} + B_0 \log n \cdot \log(1/\varepsilon)$, for $A_0 = B_0 = 5000000$. This will prove the theorem for $\varepsilon < 1/100$. To prove the theorem for all $\varepsilon \le 1/3$, we use error reduction (Fact 6) and reduce the error to $1/100$ and then apply the result for small error.

The proof is by induction on the parameters n, t and ε. At any stage of the induction, given an (n,t)-threshold tuple with error parameter ε, we construct the required probabilistic poly-tuple by using the probabilistic poly-tuples (guaranteed by inductive hypothesis) for suitable threshold poly-tuples with $n/10$ inputs and error parameter $\varepsilon/4$. Thus the base cases of the induction are as follows.

Base Case: Suppose $n \leq 10$. Let $T = (T_1, \ldots, T_m)$ be an (n,t)-threshold tuple. Let $Q_1, \ldots, Q_m$ be the unique multilinear polynomial representations of $T_1, \ldots, T_m$ respectively. Then $Q = (Q_1, \ldots, Q_m)$ is an ε-error probabilistic poly-tuple for T, for all $\varepsilon \in (0, 1/100)$, with $\deg Q \leq n = 10$.

Base Case: Suppose $\varepsilon \leq 2^{-t/160000}$. Let $T = (T_1, \ldots, T_m) = (\mathrm{Thr}_n^{t_1}, \ldots, \mathrm{Thr}_n^{t_m})$ be any (n,t)-threshold tuple and let $r = 160000 \log(1/\varepsilon)$.

Suppose $n \leq r$. Let $Q_1, \ldots, Q_m$ be the unique multilinear representations of $T_1, \ldots, T_m$ respectively. Then $Q = (Q_1, \ldots, Q_m)$ is an ε-error probabilistic polynomial with $\deg Q \leq n \leq r = \lceil \log(1/\varepsilon) \rceil$. Now suppose $n > r$. Let $P_1 = (\mathrm{EX}_{[0,r]} T_1, \ldots, \mathrm{EX}_{[0,r]} T_m)$. Then $\deg P_1 \leq r$. Choose a uniformly random hash function $\mathbf{H} : [n] \to [r]$ and let $\mathbf{S}_j = \mathbf{H}^{-1}(j)$, $j \in [r]$.

First let us suppose that $\mathrm{char}(\mathbb{F}) = p > 0$. Choose $\alpha_i \sim \mathbb{F}_p$, $i \in [n]$ independently and uniformly at random and define $\mathbf{L}_j(x) = \sum_{i \in \mathbf{S}_j} \alpha_i x_i$, for $x \in \{0,1\}^n$, $j \in [r]$. For $i \in [m]$, let $\mathbf{P}_2^{(i)} = Q_r^{(i)}(\mathbf{L}_1^{p-1}, \ldots, \mathbf{L}_r^{p-1})$, where $Q_r^{(i)}$ is the unique multilinear polynomial representation of $\mathrm{Thr}_r^{t_i}$. Let $\mathbf{P}_2 = (\mathbf{P}_2^{(1)}, \ldots, \mathbf{P}_2^{(m)})$. Define $\mathbf{P} = \mathbf{1} - (\mathbf{1} - P_1) * (\mathbf{1} - \mathbf{P}_2)$, that is, $\mathbf{P} = (\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(m)})$, where $\mathbf{P}^{(i)} = \mathrm{OR}_2(P_1^{(i)}, \mathbf{P}_2^{(i)})$, for all $i \in [m]$.

Note that since $\varepsilon \leq 2^{-t/160000}$, we have $r = 4800000 \log(1/\varepsilon) \geq t$. Thus $t_i \leq t \leq r$, for all $i \in [m]$. Now fix any $a \in \{0,1\}^n$. Let $Z_a = \{i \in [m] : \mathrm{Thr}_n^{t_i}(a) = 0\}$ and $N_a = \{i \in [m] : \mathrm{Thr}_n^{t_i}(a) = 1\}$. So we have $|a| < t_i \leq t \leq r$ and hence $\mathrm{EX}_{[0,r]} T_i(a) = 0$, for all $i \in Z_a$. Also $|(\mathbf{L}_1^{p-1}(a), \ldots, \mathbf{L}_r^{p-1}(a))| \leq |a| < t_i$ w.p.1, and so $\mathbf{P}_2^{(i)}(a) = Q_r^{(i)}((\mathbf{L}_1^{p-1}(a), \ldots, \mathbf{L}_r^{p-1}(a))) = 0$ w.p.1, for all $i \in Z_a$ simultaneously. Thus $\mathbf{P}^{(i)}(a) = 0$ w.p.1, for all $i \in Z_a$ simultaneously.

Further we have $|a| \geq t_i$, for all $i \in N_a$. We will now show that $\mathbf{P}^{(i)}(a) = 1$ w.p. at least $1 - \varepsilon$, for all $i \in N_a$ simultaneously. If $|a| \leq r$, then again $P_1^{(i)}(a) = 1$, for all $i \in N_a$ and so $\mathbf{P}^{(i)}(a) = 1$ w.p.1. Now suppose $|a| \geq r$. Without loss of generality, assume $t_1 \leq \cdots \leq t_m = t$. Then we have $\mathbf{P}_2^{(1)}(a) \geq \cdots \geq \mathbf{P}_2^{(m)}(a)$ w.p.1, under the order $1 > 0$. So it is enough to show that $\mathbf{P}^{(m)}(a) = 1$ w.p. at least $1 - \varepsilon$.

Define $I(\mathbf{H}) = \{j \in [r] : \mathrm{supp}(a) \cap \mathbf{S}_j \neq \emptyset\}$. We get

$$\begin{aligned}
\Pr\left[\mathbf{P}_2^{(m)}(a) = 0\right] &= \Pr\left[\mathbf{P}_2^{(m)}(a) = 0 \mid |I(\mathbf{H})| < r/10\right] \cdot \Pr\left[|I(\mathbf{H})| < r/10\right] \\
&\quad + \Pr\left[\mathbf{P}_2^{(m)}(a) = 0 \mid |I(\mathbf{H})| \geq r/10\right] \cdot \Pr\left[|I(\mathbf{H})| \geq r/10\right] \\
&\leq \Pr\left[|I(\mathbf{H})| < r/10\right] + \max_{\mathbf{H} : |I(\mathbf{H})| \geq r/10} \Pr\left[\mathbf{P}_2^{(m)}(a) = 0 \mid \mathbf{H}\right].
\end{aligned}$$

By Union Bound, we get

$$\Pr\left[|I(\mathbf{H})| < r/10\right] \leq \sum_{I \subseteq [r],\, |I| = r/10} \Pr\left[I(\mathbf{H}) \subset I\right] \leq \binom{r}{r/10} \frac{1}{10^r} \leq \frac{1}{4^r} \leq \frac{1}{4} \cdot \frac{1}{2^r} \leq \frac{\varepsilon}{4}.$$

Now fix any $\mathbf{H}$ such that $|I(\mathbf{H})| \geq r/10$, and let $\ell = |I(\mathbf{H})|$. Note that $\mathbf{P}_2^{(m)}(a)$ is 0 if and only if at most $t-1$ many $\mathbf{L}_j(a)$ are non-zero. We consider only $j \in I(\mathbf{H})$. For each $j \in I(\mathbf{H})$, the probability that $\mathbf{L}_j(a)$ is non-zero is $1 - 1/p \geq 1/2$. Thus, the expected number of $\mathbf{L}_j(a)$ $(j \in I(\mathbf{H}))$ that are non-zero is at least $\ell/2 \geq r/20$. Thus, by Lemma 21,

$$\Pr\left[\mathbf{P}_2^{(m)}(a) = 0 \mid \mathbf{H}\right] = \Pr\left[|I(\mathbf{H}) \cap \{j : \mathbf{L}_j(a) = 1\}| \leq t - 1 \mid \mathbf{H}\right] \leq 2\exp\left(-\frac{r}{240}\right) < \frac{\varepsilon}{2}.$$

where for the inequality we have used the fact that $t \leq r/40$. Thus $\Pr\left[\mathbf{P}_2^{(m)}(a) = 0\right] \leq \varepsilon$, proving the base case when $\mathrm{char}(\mathbb{F}) = p > 0$.

Now suppose $\mathrm{char}(\mathbb{F}) = 0$. Then for $i \in [m]$ we let $\mathbf{P}_2^{(i)} = Q_r^{(i)}(\mathbf{O}_1, \ldots, \mathbf{O}_r)$, where $Q_r^{(i)}$ is the unique multilinear polynomial representation of $\mathrm{Thr}_r^{t_i}$, and for $j \in [r]$, $\mathbf{O}_j$ is a 1/3-error probabilistic polynomial for $\mathrm{OR}_{\mathbf{S}_j}$, the OR function on variables $(X_k : k \in \mathbf{S}_j)$. Let $\mathbf{P}_2 = (\mathbf{P}_2^{(1)}, \ldots, \mathbf{P}_2^{(m)})$. Define $\mathbf{P} = \mathbf{1} - (\mathbf{1} - P_1) * (\mathbf{1} - \mathbf{P}_2)$, that is, $\mathbf{P} = (\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(m)})$, where $\mathbf{P}^{(i)} = \mathrm{OR}_2(P_1^{(i)}, \mathbf{P}_2^{(i)})$, for all $i \in [m]$. The rest of the analysis follows similarly, proving the base case when $\mathrm{char}(\mathbb{F}) = 0$.

Inductive Construction. For any positive characteristic p, any $n' < n$, $t' \in [0, n']$ and $\varepsilon' \in (0, 1/100)$, assume the existence of an ε'-error probabilistic poly-tuple for any (n', t')-threshold tuple, with degree at most $A_p\sqrt{t' \log(1/\varepsilon')} + B_p \log(1/\varepsilon')$; similarly, for characteristic zero, assume we have a probabilistic poly-tuple of degree $A_0 \log n \cdot \sqrt{t' \log(1/\varepsilon')} + B_0 \log n \cdot \log(1/\varepsilon')$.

We now consider an (n, t)-threshold tuple $T = (T_1, \ldots, T_m) = (\mathrm{Thr}_n^{t_1}, \ldots, \mathrm{Thr}_n^{t_m})$. Assume that the parameter $\varepsilon > 2^{-t/160000}$ since otherwise can use the construction from the base case. Define

$$T' = (T'_1, \ldots, T'_m) = \left(\mathrm{Thr}_{n/10}^{t_1/10}, \ldots, \mathrm{Thr}_{n/10}^{t_m/10}\right),$$
$$T''_+ = (T''_{1,+}, \ldots, T''_{m,+}) = \left(\mathrm{Thr}_{n/10}^{t_1/10 + 20\sqrt{t\log(1/\varepsilon)}}, \ldots, \mathrm{Thr}_{n/10}^{t_m/10 + 20\sqrt{t\log(1/\varepsilon)}}\right),$$
$$T''_- = (T''_{1,-}, \ldots, T''_{m,-}) = \left(\mathrm{Thr}_{n/10}^{t_1/10 - 20\sqrt{t\log(1/\varepsilon)}}, \ldots, \mathrm{Thr}_{n/10}^{t_m/10 - 20\sqrt{t\log(1/\varepsilon)}}\right).$$

By induction hypothesis, let $\mathbf{T}', \mathbf{T}''_+, \mathbf{T}''_-$ be $\varepsilon/4$-error probabilistic poly-tuples for T', T''_+, T''_- respectively. Let $\mathbf{N}'' = (\mathbf{1} - \mathbf{T}''_+) * \mathbf{T}''_-$. For any $x \in \{0,1\}^n$, choose a random subvector $\hat{\mathbf{x}} \in \{0,1\}^{n/10}$ with each coordinate chosen independently with probability 1/10, with replacement. Define

$$\mathbf{T}(x) = \mathbf{N}''(\hat{\mathbf{x}}) * E(x) + (\mathbf{1} - \mathbf{N}'')(\hat{\mathbf{x}}) * \mathbf{T}'(\hat{\mathbf{x}}),$$

where $E = (E_1, \ldots, E_m)$, with $E_i = \mathrm{EX}_{[t_i - 300\sqrt{t\log(1/\varepsilon)}, t_i + 300\sqrt{t\log(1/\varepsilon)}]}\mathrm{Thr}_n^{t_i}$, $i \in [m]$. We will now prove that $\mathbf{T}$ is an ε-error probabilistic poly-tuple for T.

Correctness of Inductive Construction. We now check that the construction above gives an ε-error probabilistic poly-tuple for T. Fix any $a \in \{0,1\}^n$. Let $\hat{\mathbf{a}} \in \{0,1\}^{n/10}$ be chosen as given in the inductive construction.

Suppose $|a| \leq 2t$. Let $\theta = 10\sqrt{t\log(1/\varepsilon)}$. Applying Lemma 21, we get $\Pr\left[||\hat{\mathbf{a}}| - |a|/10| > \theta\right] < \varepsilon/4$. By induction hypothesis, the probability that $\mathbf{T}'(\hat{\mathbf{a}})$ does not agree with $T'(\hat{\mathbf{a}})$ is at most $\varepsilon/4$, and similarly for $\mathbf{T}''_+$ and $\mathbf{T}''_-$. Let $\mathcal{G}_a$ be the event that none of the above events occur; by a union bound, the event $\mathcal{G}_a$ occurs with probability at least $1 - \varepsilon$. In this case, we show that $\mathbf{T}(a) = T(a)$, which will prove the correctness of the construction in the case that $|a| \leq 2t$.

To see the above, observe the following for each $i \in [m]$.

- $\mathbf{T}'_i(\hat{\mathbf{a}}) = T_i(a)$ if $||a| - t_i| > 10\theta$. This is because $\mathbf{T}'_i(\hat{\mathbf{a}}) = T'_i(\hat{\mathbf{a}})$ by our assumption that the event $\mathcal{G}_a$ has occurred. Further, we also have $T'_i(\hat{\mathbf{a}}) = T_i(a)$ since $|\hat{a} - |a|/10| \leq \theta$ (by occurrence of $\mathcal{G}_a$) and hence $|a| \geq t_i$ if and only if $|\hat{a}| \geq t_i/10$.

- If $||a| - t_i| > 30\theta$, then $\mathbf{N}_i''(\hat{\mathbf{a}}) = 0$. This is because $||\hat{a}| - |a|/10| \leq \theta$ and hence $||\hat{a}| - t_i/10| > 2\theta$. Hence, either $\mathbf{T}_{i,+}''(\hat{a}) = 1$ or $\mathbf{T}_{i,-}''(\hat{a}) = 0$ and therefore, $\mathbf{N}_i''(\hat{\mathbf{a}}) = 0$. Thus, when $||a| - t_i| > 30\theta$, the definition of $\mathbf{T}$ yields $\mathbf{T}_i(a) = \mathbf{T}_i'(\hat{\mathbf{a}}) = T_i(a)$. We are therefore done in this case.
- If $||a| - t_i| < 10\theta$, then $\mathbf{N}_i''(\hat{\mathbf{a}}) = 1$. This is similar to the analogous statement above. Therefore, when $||a| - t_i| < 10\theta$, we have $\mathbf{T}_i(a) = E_i(a) = T_i(a)$ as $|a| \in [t_i - 300\sqrt{t\log(1/\varepsilon)}, t_i + 300\sqrt{t\log(1/\varepsilon)}]$. Hence, we are done in this case also.
- If $10\theta \leq ||a| - t_i| \leq 30\theta$, then $E_i(a) = \mathbf{T}'(\hat{\mathbf{a}}) = T_i(a)$. Since $\mathbf{N}_i''(\hat{\mathbf{a}}) \in \{0, 1\}$ for each $i \in [m]$, we again obtain $\mathbf{T}_i(a) = T_i(a)$.

This shows that for any a such that $|a| \leq 2t$, whenever $\mathcal{G}_a$ does not occur, $\mathbf{T}(a) = T(a)$.

Now suppose $|a| > 2t$. Then by a Chernoff bound (follows from Lemma 21), we get $\Pr[|\hat{\mathbf{a}}| < 1.5t/10] < 2\exp(-t/400) < \varepsilon/2$. Also, by the induction hypothesis, the probability that $\mathbf{T}'(\hat{\mathbf{a}})$ does not agree with $T'(\hat{\mathbf{a}})$ is at most $\varepsilon/4$, and similarly for $\mathbf{T}_+''$ and $\mathbf{T}_-''$. Let $\mathcal{G}_a$ denote the event that none of the above events occur; we have $\Pr[\mathcal{G}] \geq 1 - \varepsilon$. As above, we show that when $\mathcal{G}_a$ occurs, then $\mathbf{T}(a) = T(a)$.

To see this, we proceed as follows.

- Since $|a| \geq 2t$ and $|\hat{\mathbf{a}}| \geq 1.5t/10$, both $T(a)$ and $\mathbf{T}_i'(\hat{a})$ are both the constant-1 vector.
- Further, we note that we have $\mathbf{N}_i''(\hat{\mathbf{a}}) = 0$ for each $i \in [m]$. This is because $||\hat{\mathbf{a}}| - t_i/10| \geq (|\hat{\mathbf{a}}| - t/10) \geq t/20 > 20\sqrt{t\log(1/\varepsilon)}$.

This implies that $\mathbf{T}_i(a) = \mathbf{T}_i'(\hat{\mathbf{a}}) = 1$ for each $i \in [m]$.

Hence, when $\mathcal{G}_a$ does not occur, we have $\mathbf{T}(a) = T(a)$, which proves the correctness of the construction.

Correctness of Degree. The computation that shows that $\deg(\mathbf{T})$ satisfies the inductive claim is omitted here and is in the full version of the paper. ◀

3.2 Upper bounds from Theorem 3

Upper bound for $\mathrm{pdeg}_\varepsilon(g)$. This result is due to Lu [12].

Upper bound for $\mathrm{pdeg}_\varepsilon(h)$. Let $B(h) = k$. Thus we can write $h = h_1 + (1 - \widetilde{h_2})$, for k-constant symmetric Boolean functions h_1, h_2, where $\widetilde{h_2}(x_1, \ldots, x_n) = h_2(1 - x_1, \ldots, 1 - x_n)$. But then by Corollary 20, $\mathrm{pdeg}_\varepsilon(h_1) = \mathrm{pdeg}_\varepsilon(h_2) = O(\sqrt{k\log(1/\varepsilon)} + \log(1/\varepsilon))$ and so $\mathrm{pdeg}_\varepsilon(h) = O(\sqrt{k\log(1/\varepsilon)} + \log(1/\varepsilon))$ over positive characteristic p. For $p = 0$, we obtain the same upper bound up to log-factors.

Upper bound for $\mathrm{pdeg}_\varepsilon(f)$. Let (g, h) be the standard decomposition of f. So $f = g \oplus h = g + h - 2gh$. Further, we already have the Alman-Williams bound of $O(\sqrt{n\log(1/\varepsilon)})$ on $\mathrm{pdeg}_\varepsilon(f)$ (Lemma 11). So we get $\mathrm{pdeg}_\varepsilon(f) = O(\min\{\sqrt{n\log(1/\varepsilon)}, \mathrm{per}(g) + \sqrt{B(h)\log(1/\varepsilon)} + \log(1/\varepsilon)\})$ over positive characteristic and the same bound up to log-factors over characteristic 0. This concludes the proof of Theorem 3.

4 Lower bounds

In this section, we prove the lower bounds from Theorem 4.

Throughout, $\mathbb{F}$ is fixed to be some arbitrary field of characteristic p (possibly 0). We use $\mathrm{pdeg}_\varepsilon(\cdot)$ instead of $\mathrm{pdeg}_\varepsilon^{\mathbb{F}}(\cdot)$ and $\mathrm{pdeg}(\cdot)$ instead of $\mathrm{pdeg}_{1/3}^{\mathbb{F}}(\cdot)$.

4.1 Preliminary lemmas

We need some preliminary lemmas. The proofs are omitted for lack of space.

▶ **Lemma 23.** *Let $n, m \in \mathbb{N}$, $n > m$ and $\varepsilon \leq 1/3$. Let $f \in s\mathcal{B}_n$ and $g \in s\mathcal{B}_m$ be such that* $\mathrm{per}(g)$ *divides* $\mathrm{per}(f)$ *and* $\mathrm{per}(f) \leq (n-m+2)/2$. *Then* $\mathrm{pdeg}_\varepsilon(f) = \Omega(\mathrm{pdeg}_\varepsilon(g)/\log^3(n/\varepsilon))$.

▶ **Lemma 24.** *For any* $1/2^n \leq \varepsilon \leq 1/3$, $\mathrm{pdeg}_\varepsilon(\mathrm{EThr}_n^{\lceil n/2\rceil}) = \widetilde{\Omega}(\sqrt{n\log(1/\varepsilon)})$.

4.2 Lower bounds from Theorem 4

We recall the lower bound for periodic functions from Theorem 4. In light of Observation 2, this is a slightly more general statement.

▶ **Lemma 25.** *Let $g \in s\mathcal{B}_n$ be any function with* $\mathrm{per}(g) = b \leq n/3$, *then* $\mathrm{pdeg}_\varepsilon(g) = \widetilde{\Omega}(\sqrt{n})$ *if* $\mathrm{per}(g) > 1$ *and is not a power of p and* $\widetilde{\Omega}(\min\{\sqrt{n}, \mathrm{per}(g)\})$ *otherwise.*

Proof. Assume $\mathrm{per}(g) = b > 1$. Consider the two cases below.

b is not a power of p. Let b' be any non-trivial divisor of b which is coprime to p (if $p = 0$, we simply take $b' = b$). For $i \in [0, b'-1]$, define $g_i = \mathrm{MOD}^{b',i}_{\lceil n/3\rceil}$. The functions g and g_i satisfy the hypotheses of Lemma 23 and therefore for any constant $\varepsilon \leq 1/3$, $\mathrm{pdeg}_\varepsilon(g) = \widetilde{\Omega}(\mathrm{pdeg}_\varepsilon(g_i))$.

Note that as $b \leq n/3$, we have $b' \leq n/6 \leq \frac{1}{2}\lceil n/3\rceil$. Hence, by Lemma 9, for some $i \in [0, b'-1]$, $\mathrm{pdeg}_{1/n^2}(g_i) = \Omega(\sqrt{n\log(n^2/b)}) = \widetilde{\Omega}(\sqrt{n})$.

Therefore $\mathrm{pdeg}_{1/n^2}(g) = \widetilde{\Omega}(\sqrt{n})$ and hence by Fact 6 item 1 $\mathrm{pdeg}(g) = \widetilde{\Omega}(\sqrt{n})$.

$b = p^k$ for some $k \in \mathbb{N}$. Let $m = \min(b^2/100, \lceil n/3\rceil)$. Let $g' \in s\mathcal{B}_m$ with $\mathrm{per}(g') = b$ be such that $\mathrm{Spec}\,(g')(i) = 0$ whenever $\lfloor m/2\rfloor - \lfloor b/2\rfloor \leq i \leq \lfloor m/2\rfloor$ and $\mathrm{Spec}\,(g')(i) = 1$ whenever $\lfloor m/2\rfloor < i \leq \lfloor m/2\rfloor + b - \lfloor b/2\rfloor - 1$.

Again, the functions g and g' satisfy the hypotheses of Lemma 23 and therefore for any constant $\varepsilon \leq 1/3$, $\mathrm{pdeg}_\varepsilon(g) = \widetilde{\Omega}(\mathrm{pdeg}_\varepsilon(g'))$.

Note that g' agrees with the Maj_m function on all inputs $x \in \{0,1\}^m$ such that $||x| - (m/2)|$ is at most $b/2$. By a Chernoff bound (follows from Lemma 21),

$$\Pr_{x\in\{0,1\}^m}[||x| - m/2| > b/2] \leq 2e^{-\frac{b^2}{2m}} = 2e^{-50} < 1/5.$$

Therefore g' agrees with Maj_m on more than $4/5$ fraction of inputs and hence by Lemma 8, $\mathrm{pdeg}(g') = \Omega(\sqrt{m})$. Therefore, $\mathrm{pdeg}(g) = \widetilde{\Omega}(\sqrt{m}) = \min(\widetilde{\Omega}(b), \widetilde{\Omega}(\sqrt{n}))$. ◀

We now recall the lower bound for bounded symmetric Boolean functions from Theorem 4.

▶ **Lemma 26.** *Let $h \in s\mathcal{B}_n$ be such that $B(h) \leq \lceil n/3\rceil$. Then,* $\mathrm{pdeg}_\varepsilon(h) = \widetilde{\Omega}(\sqrt{B(h)})$,

Proof. Let $B(h) = b$. Then, we know that $\mathrm{Spec}\,h(i) = 0$ for $i \in [b, n-b]$ and further either $\mathrm{Spec}\,h(b-1)$ or $\mathrm{Spec}\,h(n-b+1)$ is 1. We assume w.l.o.g. that $\mathrm{Spec}\,h(n-b+1) = 1$ (the other case is similar).

Fix some integer $b' = b - O(1)$ so that $2b + 2\lfloor b'/2\rfloor \leq n$. Define $h' \in s\mathcal{B}_{b'}$ as

$$h'(x) = \bigvee_{i\in[0,\lfloor b'/2\rfloor]} h(x1^{n-b-2\lfloor b'/2\rfloor+i}0^{b-b'+2\lfloor b'/2\rfloor-i}), \quad \text{for all } x \in \{0,1\}^{b'}.$$

We claim that $h' = \mathrm{Maj}_{b'}$. To show this, we proceed as follows.

Let $|x| \leq b'/2$ and therefore $|x| \leq \lfloor b'/2 \rfloor$. Then for all $i \in [0, \lfloor b'/2 \rfloor]$, using our choice of b', we have

$$b \leq n - b - 2\lfloor b'/2 \rfloor \leq |x1^{n-b-2\lfloor b'/2 \rfloor + i}0^{b-b'+2\lfloor b'/2 \rfloor - i}| = |x| + (n - b - 2\lfloor b'/2 \rfloor + i) \leq n - b$$

and therefore none of the terms in the OR above evaluate to 1. Thus $h'(x) = 0$.

Let $|x| > b'/2$ and therefore $|x| \geq \lfloor b'/2 \rfloor + 1$. Let $|x| = \lfloor b'/2 \rfloor + j$ for some $j \in [1, \lceil b'/2 \rceil]$. Let $i = \lfloor b'/2 \rfloor - j + 1$. Then $|x1^{n-b-2\lfloor b'/2 \rfloor + i}0^{b-b'+2\lfloor b'/2 \rfloor - i}| = n - b + 1$. Therefore the OR evaluates to 1 and $h'(x) = 1$.

From the above we see that $h' = \mathrm{Maj}_b$. Now,

$$\begin{aligned}
\mathrm{pdeg}(h') &\leq \mathrm{pdeg}_{2/n}(h') \\
&\leq \mathrm{pdeg}_{1/n}(\mathrm{OR}) \cdot \mathrm{pdeg}_{1/n^2}(h) \\
&\leq O(\log^2 n) \cdot O(\log n) \cdot \mathrm{pdeg}(h) \\
&\leq \widetilde{O}(\mathrm{pdeg}(h)).
\end{aligned}$$

The second inequality follows from Fact 6 item 2 and the third inequality follows from Lemma 7 and Fact 6 item 1.

Since $\mathrm{pdeg}(\mathrm{Maj}_{b'}) = \Omega(\sqrt{b'}) = \Omega(\sqrt{b})$, it follows that $\mathrm{pdeg}(h) = \widetilde{\Omega}(\sqrt{b})$. ◀

Using the above, a short case analysis yields the lower bound on $\mathrm{pdeg}(f)$ for general $f \in s\mathcal{B}_n$. The proof is omitted.

References

1 Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–150. IEEE, 2015.

2 Richard Beigel. The polynomial method in circuit complexity. *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 82–95, 1993.

3 Richard Beigel, Nick Reingold, and Daniel A. Spielman. The Perceptron Strikes Back. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference, Chicago, Illinois, USA, June 30 - July 3, 1991*, pages 286–291, 1991. doi:10.1109/SCT.1991.160270.

4 Siddharth Bhandari, Prahladh Harsha, Tulasimohan Molli, and Srikanth Srinivasan. On the Probabilistic Degree of OR over the Reals. In Sumit Ganguly and Paritosh Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2018.5.

5 Mark Braverman. Polylogarithmic independence fools AC^0 circuits. *J. ACM*, 57(5), 2010. doi:10.1145/1754399.1754401.

6 Bettina Brustmann and Ingo Wegener. The Complexity of Symmetric Functions in Bounded-Depth Circuits. *Inf. Process. Lett.*, 25(4):217–219, 1987. doi:10.1016/0020-0190(87)90163-3.

7 Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

8 Ronald Fagin, Maria M. Klawe, Nicholas Pippenger, and Larry J. Stockmeyer. Bounded-Depth, Polynomial-Size Circuits for Symmetric Functions. *Theor. Comput. Sci.*, 36:239–250, 1985. doi:10.1016/0304-3975(85)90045-3.

9 Prahladh Harsha and Srikanth Srinivasan. On Polynomial Approximations to AC^0. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*, volume 60 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.32.

10 David Lawrence Johnson, David Leroy Johnson, and SS Johnson. *Topics in the theory of group presentations*, volume 42. Cambridge University Press, 1980.

11 Nathan Linial and Noam Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365, 1990.

12 Chi-Jen Lu. An exact characterization of symmetric functions in $\mathrm{qAC}^0[2]$. *Theoretical Computer Science*, 261(2):297–303, 2001.

13 Raghu Meka, Oanh Nguyen, and Van Vu. Anti-concentration for Polynomials of Independent Random Variables. *Theory of Computing*, 12(1):1–17, 2016. doi:10.4086/toc.2016.v012a011.

14 Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, New York, NY, USA, 2014.

15 Ramamohan Paturi. On the degree of polynomials that approximate symmetric Boolean functions (preliminary version). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 468–474. ACM, 1992.

16 Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematicheskie Zametki*, 41(4):598–607, 1987. (English translation in *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987). doi:10.1007/BF01137685.

17 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82. ACM, 1987.

18 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.

19 Roman Smolensky. On representations by low-degree polynomials. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 130–138. IEEE, 1993.

20 Roman Smolensky. On Representations by Low-Degree Polynomials. In *FOCS*, pages 130–138, 1993. doi:10.1109/SFCS.1993.366874.

21 Srikanth Srinivasan, Utkarsh Tripathi, and S. Venkitesh. On the Probabilistic Degrees of Symmetric Boolean functions. *Electronic Colloquium on Computational Complexity (ECCC)*, 138:1–24, 2019. URL: https://eccc.weizmann.ac.il/report/2019/138.

22 Jun Tarui. Probabilistic Polynomials, AC^0 Functions, and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 113(1):167–183, 1993.

23 Richard Ryan Williams. The polynomial method in circuit complexity applied to algorithm design (invited talk). In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

24 Zhi-Li Zhang, David A Mix Barrington, and Jun Tarui. Computing symmetric functions with AND/OR circuits and a single MAJORITY gate. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 535–544. Springer, 1993.

Classification Among Hidden Markov Models

S. Akshay
IIT Bombay, India

Hugo Bazille
Univ Rennes, Inria, IRISA, France

Eric Fabre
Univ Rennes, Inria, IRISA, France

Blaise Genest
Univ Rennes, CNRS, IRISA, France

Abstract

An important task in AI is one of classifying an observation as belonging to one class among several (e.g. image classification). We revisit this problem in a verification context: given k partially observable systems modeled as Hidden Markov Models (also called labeled Markov chains), and an execution of one of them, can we eventually classify which system performed this execution, just by looking at its observations? Interestingly, this problem generalizes several problems in verification and control, such as fault diagnosis and opacity. Also, classification has strong connections with different notions of distances between stochastic models.

In this paper, we study a general and practical notion of classifiers, namely *limit-sure classifiers*, which allow misclassification, i.e. errors in classification, as long as the probability of misclassification tends to 0 as the length of the observation grows. To study the complexity of several notions of classification, we develop techniques based on a simple but powerful notion of stationary distributions for HMMs. We prove that one cannot classify among HMMs iff there is a *finite separating word* from their stationary distributions. This provides a direct proof that classifiability can be checked in PTIME, as an alternative to existing proofs using *separating events* (i.e. *sets of* infinite separating words) for the total variation distance. Our approach also allows us to introduce and tackle new notions of classifiability which are applicable in a security context.

2012 ACM Subject Classification Theory of computation

Keywords and phrases verification: probabilistic systems, partially observable systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.29

Related Version A full version of the paper is available at `http://perso.crans.org/genest/abfg19.pdf`.

Funding This work has been partially supported by DST/CEFIPRA/INRIA Associated team EQuaVE and DST Inspire Faculty Award [IFA12-MA-17].

Acknowledgements We would like to thank Stefan Kiefer for his expert opinion, as well as anonymous reviewers for their constructive comments.

1 Introduction

The spectacular success of artificial intelligence (AI) and machine learning techniques in many varied application domains in the last decade has led to the emergence of several new and old questions, especially regarding their guarantees and correctness. This has led to several recent projects at the interface of formal methods and AI, whose broad goal is to formally reason and verify properties about these AI models and tasks. One such important task in AI is classification, which is a fundamental problem with many practical applications,

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 29; pp. 29:1–29:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

e.g., in image processing. In this paper, we consider classification in a verification context. One main issue when verifying systems is partial observability. It is thus important to know what information can be recovered from a partially observable system.

We first consider a system perspective: we want to know whether, no matter the execution of the system, some hidden information is retrievable, at least with high probability. To represent the system, we thus consider a partially observable stochastic model, namely *Hidden Markov Models (HMM for short)* [12, 10], also known as labeled Markov chains [6] or probabilistic labeled transition systems [5]. While notationally different, these various models are equivalent in terms of expressive power. In HMMs, states are not observable, but we get some (potentially stochastic) signals from states. In the specific variant of HMMs that we study in this paper, we encode the signals from states as labels of transitions exiting states. That is, the observation from an execution of an HMM is its labeling sequence. We encode the different hidden information as several HMMs, with different transition probabilities. Finding the hidden information from the observation thus amounts to classifying the observation among the different HMMs.

Many problems concerning systems with hidden information can be recast in the framework of classification, such as, (i) fault diagnosis: classifying between a faulty system that has executed errors and the system without faults [14, 15, 4, 5]; (ii) opacity: classifying between high and low privilege parts of the system [10], etc. Although some problems are incomparable (e.g. diagnosis is intrinsically "asymmetric" while classification is "symmetric"), most proof techniques and ideas are common. Moreover, results on classification problems have been applied to show results in these related contexts. While it is not our aim to survey these applications here, we provide two instances: a fault diagnosis problem [5] is solved using a result on distance between stochastic systems [6], which is equivalent with classification [11]. Also, opacity is cast as a classification problem in [10]. We hence believe that classification is a good framework to state and prove algorithmic and complexity results.

Several notions of classification can be defined: sure, almost-sure, and limit-sure, depending respectively on whether we want the classification to eventually happen for sure, with probability 1, or with arbitrarily small error. The first two notions have classical solutions coming from fault-diagnosis [14, 4]: the existence of such classifiers can be checked in PTIME and PSPACE respectively. The third notion is however the most practical as the classifier is the most powerful: it can use the long run statistics on observations to take its decision (e.g. the frequency of *ab*'s in the word). It is also the hardest notion to study for this very reason.

We focus on this notion of limit-sure classification in this paper. First, a closely-related problem of *distinguishability* has been proved to be in PTIME by [11], using the PTIME algorithm from [6] to test whether the total variation metric between two HMMs is 1. We reinvestigate these deep results using different techniques, which shines some new light on this problem. Our starting point is the following: for a very restricted class of HMMs [10], whose underlying Markov chains are ergodic and crucially, assuming that initial distributions have non-zero probability on every state, it is sufficient to consider the statistics on *states* (e.g. the frequency of state s). These statistics on *states* are obtained by [10] using the classical notion of stationary distributions over the underlying Markov Chain, i.e. the HMM where we forget all observations. As we show in Example 2, stationary distributions on Markov chains do not suffice for solving limit-sure classification for general HMMs. We build on this idea and propose a new notion to study the long run statistics of the *observations*.

Our first contribution is to develop the notion of stationary distributions for *general HMMs* to study the long run statistics of the observations. To do so, we focus on beliefs, that is, the set of states that can be reached with the same observation. We show that a notion of stationary distributions can be defined for beliefs in Bottom Strongly Connected Components (BSCCs), and that it also corresponds to a notion of asymptotic distributions, describing the asymptotic statistics of beliefs. This generalizes stationary distributions for Markov chains: for instance, irreducible Markov chains of period k correspond to cycling through k different beliefs. We believe that this notion can find applications in other contexts.

Our next contribution is to show how this notion of stationary distribution of HMMs can be used to characterize limit-sure classifiability. We show that we cannot classify between HMMs iff they have beliefs which can be reached by the same observation and for which the stationary distributions can be separated by *one finite word* (for which the probability is different). This provides a PTIME algorithm to test for limit-sure classifiability. Note that the existence of such a PTIME algorithm has been established in [11], where this result was formulated in terms of HMMs distinguishability. The proofs are different however, as [11] focuses on separating events [6], that is *sets of infinite words* with probability 0 (resp. 1) in one of the HMMs (resp. the other one), while considering stationary distributions allows us to focus on a single finite separating word with probability p (resp. $q \neq p$).

Our final contribution is to study classifiability in a security context: an attacker has different attacks against different HMMs. To be able to perform his attack, he needs to find one execution that can be classified (and thus attacked) rather than whether every execution can be classified. We call this notion *attack-classification.* We study limit-sure attack-classification using the notion of stationary distributions for HMMs developed above. We show that deciding whether there exists a limit-sure attack-classifier between two HMMs is PSPACE-complete. On the other hand, if we consider a variation on the notion of limit-sure attack-classifier, which extends distinguishability for HMMs [11], we are able to show that it is not only different from limit-sure attack-classifier, but this problem is also undecidable. All missing proofs and details can be found in the long version [1].

2 Preliminaries and Problem Statement

A *Hidden Markov Model* [12, 13, 10] (*HMM* for short) $\mathcal{A}$ on finite alphabet Σ is a tuple $\mathcal{A} = (S, M, \sigma_0)$ with S a set of states, σ_0 an initial distribution, $M : S \times \Sigma \times S \to [0; 1]$, such that for all s, $\sum_{a,s'} M(s, a, s') = 1$. Notice that this notion has been referred to using different names in the literature: labeled Markov chains, pLTS (probabilistic transition systems) in [5], probabilistic automata (not to be confused with Rabin's Probabilistic automata), etc. Classical Markov chains can be viewed as HMMs with a single letter alphabet. In what follows we assume knowledge of classical properties, definitions about Markov chains, such as irreducibility, aperiodicity and refer to [9] for a formal treatment.

A run ρ of $\mathcal{A}$ is a sequence in $S(\Sigma \times S)^*$. It starts in $s^-(\rho)$, with $\sigma_0(s^-(\rho)) > 0$, and ends in state $s^+(\rho)$. An observation w from $\mathcal{A}$ is a sequence of letters $w = a_1 \cdots a_n \in \Sigma^*$ such that there exists a run ρ made of $n+1$ states $\rho = s_0, a_1 \dots, a_n s_n$ with $\sigma_0(s_0) > 0$ and for all $i > 0$, $M(s_{i-1}, a_i, s_i) > 0$. We denote $obs(\rho) = w$. For a run $\rho = s_0, a_1 \dots, a_n s_n$, we define its probability as $P(\rho) = \sigma_0(s_0) \cdot \prod_{i=1}^{n} M(s_{i-1}, a_i, s_i)$. We sometimes abuse notation and write $M(s_1, w, s_n)$ to mean $\prod_{i=1}^{n} M(s_{i-1}, a_i, s_i)$. We define the probability of an observation $w \in \Sigma^*$ as $P(w) = \sum_{\rho | obs(\rho)=w} P(\rho)$. In general we write $P_\sigma^{\mathcal{A}}$ to express the probability in HMM $\mathcal{A}$ with initial distribution σ. If $\sigma(s) = 1$, then we use $P_s^{\mathcal{A}}$ instead.

A *non-deterministic finite automaton (NFA for short)* is as usual a structure $\mathcal{A} = (S, \Delta, S_0)$, where the transition probabilities (as in a HMM) are replaced with a transition relation Δ and initial distribution is replaced by a set of initial states S_0. For a HMM (S, M, σ_0), we can associate the NFA $\mathcal{A} = (S, \Delta, S_0)$, by taking $(s, a, t) \in \Delta$ iff $M(s, a, t) > 0$ and $s \in S_0$ iff $\sigma_0(s) > 0$. The notion of paths and observation is preserved. Fig. 1 shows an HMM on the left and an NFA on the right.

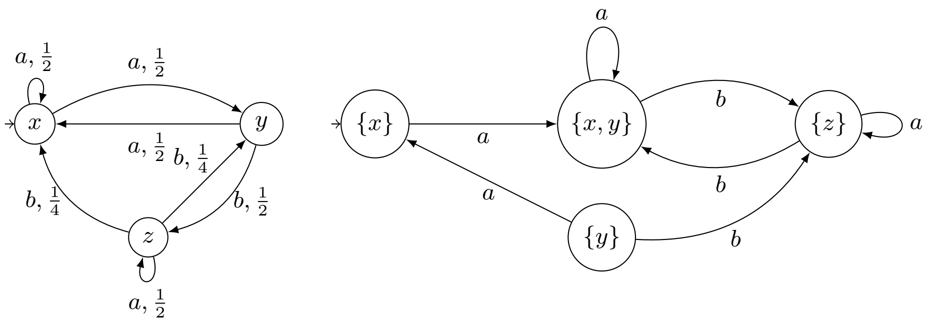


Figure 1 Example of an HMM $\mathcal{A}$ on alphabet $\Sigma = \{a, b\}$ and of an NFA $\mathcal{B}_\mathcal{A}$ on alphabet Σ.

The language of an automaton (or by extension of an HMM) is the set of observations $L(\mathcal{A}) = \{w \mid w = obs(\rho), \rho \text{ a path of } \mathcal{A}\}$. We denote by $L^\infty(\mathcal{A})$ the set of infinite observations in $\mathcal{A}$, that is such that every of its prefix is in $L(\mathcal{A})$. Finally, we use the standard way to extend probabilities to some sets of infinite paths, by means of cylinder-sets [2]. In particular, taking two HMMs $\mathcal{A}_1, \mathcal{A}_2$ on the same alphabet, $L^\infty(\mathcal{A}_1) \cap L^\infty(\mathcal{A}_2)$ is measurable. We write $L(\mathcal{A}, s)$ for the language of $\mathcal{A}$ starting in state s.

A strongly connected component C of an HMM $\mathcal{A}$ is a maximal set of states such that there is a path from any state of C to any state of C. A strongly connected component C is called a *bottom strongly connected component(BSCC)* if the only states reachable from C are in C. For instance, there is only one BSCC in the NFA of Fig. 1, with 2 states $\{x, y\}$ and $\{z\}$. Runs of an HMM end up in one of the BSCCs with probability 1.

Probabilistic Finite Automata (PFA). Several lower bounds will come from results on Rabin's *probabilistic finite automaton (PFA)* [8]. A PFA $\mathcal{A}$ on finite alphabet Σ is a tuple $\mathcal{A} = (S, (M_a)_{a\in\Sigma}, \sigma_0)$ with S a set of states, σ_0 an initial distribution, $M_a : S \times S \to [0, 1]$ for each $a \in \Sigma$, such that for all a, s, $\sum_{s'} M_a(s, s') = 1$. Similar to HMMs, the states of a PFA are not observed, but only letters $a \in \Sigma$ are. The difference is that we can control a PFA by choosing an action $a \in \Sigma$, while in HMMs, we observe passively an observation $a \in \Sigma$.

2.1 Probabilistic equivalence can be checked in PTIME

The PTIME algorithm for probabilistic equivalence is at the core of the PTIME algorithms from [6] (and hence [11, 5] using it), [10] and ours. Let σ_1, σ_2 be distributions over states of HMMs $\mathcal{A}_1, \mathcal{A}_2$ respectively. HMMs $\mathcal{A}_1, \mathcal{A}_2$ are equivalent from distributions σ_1, σ_2, denoted $(\mathcal{A}_1, \sigma_1) \equiv (\mathcal{A}_2, \sigma_2)$, if for any observation $w \in \Sigma^*$, we have $P^{\mathcal{A}_1}_{\sigma_1}(w) = P^{\mathcal{A}_2}_{\sigma_2}(w)$. In [3] (see also [6]), it is shown how to test in polynomial time whether $P^{\mathcal{A}_1}_{\sigma_1} \equiv P^{\mathcal{A}_2}_{\sigma_2}$, i.e.

$$\forall w \in \Sigma^*, \quad (\sigma_1 \quad \sigma_2) \cdot \begin{bmatrix} M_1(w) & \emptyset \\ \emptyset & M_2(w) \end{bmatrix} \cdot (1, \cdots, 1, -1, \cdots, -1)^T = 0$$

As the dimension of $Eq(\mathcal{A}_1, \mathcal{A}_2) = \{\begin{bmatrix} M_1(w) & \emptyset \\ \emptyset & M_2(w) \end{bmatrix} \cdot (1, \cdots, 1, -1, \cdots, -1)^T \mid w \in \Sigma^*\}$ is at most $|\mathcal{A}_1| + |\mathcal{A}_2|$, we can build a basis $v_1, \ldots v_\ell$ for $Eq(\mathcal{A}_1, \mathcal{A}_2)$ of size $\ell \leq |\mathcal{A}_1| + |\mathcal{A}_2|$. It suffices then to check whether $(\sigma_1 \quad \sigma_2) \cdot v_i = 0$ for all $i \leq \ell$.

Notice that equivalence of PFA has been known to be in PTIME for 30 years [16], before HMMs [3]. Actually, equivalences of HMMs of and PFAs are inter-reducible (one direction can be found in [7], and the other one is easy by considering the HMM associated with a PFA, which performs actions of the PFA uniformly at random).

2.2 The classification problem and its variants

Let $(A_i)_{i\leq k}$ be a set of HMMs representing different behaviors of a system under observation. The system secretly picks one HMM behavior to follow, i.e. it is a priori unknown which of the HMMs is being followed by the system. We want to *classify*, i.e. find out, which HMM behavior the system follows, only by looking at the observation $w \in \Sigma^*$. The longer we observe the system, the larger the length of the observation and better the information we have to find out the HMM. This leads us to the notion of classifiability. As it suffices to consider HMMs pairwise, we will consider in the following there is only a choice between $k = 2$ HMMs. We will denote them by $\mathcal{A}_1$, with n states, and $\mathcal{A}_2$, with m states. Formally, a classifier is a function $f : \Sigma^* \to \{\perp, 1, 2\}$ that outputs the index of the HMM from an observation, or possibly $\perp$ if it cannot conclude (yet). Consider for example $\mathcal{A}_1, \mathcal{A}_2$, both following the HMM in Figure 1, the difference being that $\mathcal{A}_1$ starts in x while $\mathcal{A}_2$ starts in z. If the observation starts with b, then we know the systems follows $\mathcal{A}_2$, because b is not possible from x. We can thus let $f(bw) = 2$. However, if the observation is ab^2a, then it could come from any $\mathcal{A}_1$ or $\mathcal{A}_2$. If the systems are probabilistically equivalent, then no matter how much we observe, we cannot classify among them. However, this is one extreme case. One can consider several notions of classifiability:

- *sure classifiability*: there exists a classifier f that eventually identifies the accurate HMM that generated w. That is, for all $w \in \Sigma^\infty$, there exists a finite prefix v of w and a classifier f for v such that $f(v) = 1$ (resp. $f(v) = 2$) iff there is no path ρ of $\mathcal{A}_2$ (resp. of $\mathcal{A}_1$) with $obs(\rho) = w$.
- *almost-sure classifiability*: there exists a classifier f that eventually identifies the accurate HMM that generated w with probability 1. This classifier cannot do errors, but there may be some infinite observation that cannot be classified, though the probability it happens should be 0 (such as tossing tail forever on a fair coin).
- *limit-sure classifiability*: there exists a classifier f that, for all $\epsilon > 0$, eventually provides the accurate HMM with probability $> 1 - \epsilon$. This is the most general notion: sure implies almost-sure implies limit-sure classifiability.

This leads to the two main questions that we are interested in, for each of the above notions: (i) how easy is it to decide if there exists a classifier? (ii) if there exists a classifier, how easy is it to produce one explicitly? For the first question, we can answer easily for the two first notions, which have been studied in different contexts.

▶ **Proposition 1** ([14, 4]). *We can surely classify among 2 HMMs iff $L^\infty(\mathcal{A}_1) \cap L^\infty(\mathcal{A}_2) = \emptyset$, and this can be checked in PTIME. We can almost-surely classify among 2 HMMs iff the set $L^\infty(\mathcal{A}_1) \cap L^\infty(\mathcal{A}_2)$ has probability 0, and this is a PSPACE-complete problem.*

For the first two notions, building the classifier is also easy: intuitively, it suffices to compute the set of states reached with the observation (called *belief* in the next section) for both HMMs. If the system is classifiable, one of these sets will eventually (almost surely with the second notion) become empty. The classifier answers the HMM with non-empty set.

Unlike the two first notions, *limit-sure classifiability* cannot be expressed in terms of the language. Indeed, it is possible to limit-surely classify among $\mathcal{A}_1, \mathcal{A}_2$, and yet $L(\mathcal{A}_1) = L(\mathcal{A}_2)$. Also, a limit-sure classifier can use statistics in order to give its estimate, which opens a lot of possibilities. Let us illustrate these:

▶ **Example 2.** Consider again $\mathcal{A}_1, \mathcal{A}_2$, where both are the HMM $\mathcal{A}$ from Fig. 1, where $\mathcal{A}_1$ starts from state x and $\mathcal{A}_2$ starts from state z. If the observation starts with b, then it is easy to conclude that the HMM is $\mathcal{A}_2$. If it starts with a, then the set of states which can be reached after observation a is $\{x, y\}$ in $\mathcal{A}_1$ and $\{z\}$ in $\mathcal{A}_2$, which are both in the BSCCs. Actually, after an even number of b's (and any number of a's), we still have $\{x, y\}$ the set of states possible in $\mathcal{A}_1$ and $\{z\}$ in $\mathcal{A}_2$. In the following section using stationary distributions on HMMs, we will show how to compute that if the HMM is $\mathcal{A}_1$, after an even number of b's, the long term average is $\frac{3}{5}$ to be in x and $\frac{2}{5}$ to be in y. From this, we deduce that the long term average is $\frac{4}{5} = \frac{3}{5}1 + \frac{2}{5}\frac{1}{2}$ to perform an a after an even number of b's. On the other hand, if the HMM is $\mathcal{A}_2$, then the state is z and we obtain the long term average $\frac{1}{2}$ to perform letter a after an even number of b's. As the observation grows, the average frequency over the observation will tend towards the long term average by law of large numbers. Thus the classifier $f(w) = 1$, if the average frequency of a's after an even number of b's observed in w is closer to $\frac{4}{5}$ than to $\frac{1}{2}$, is limit-sure. Notice that using the standard stationary distributions on *Markov chains* as in [10] only tells us that both $\mathcal{A}_1$ and $\mathcal{A}_2$ stay in long term average frequency $\frac{3}{7}$ in x, $\frac{2}{7}$ in y, and $\frac{2}{7}$ in z , and thus do $\frac{5}{7} = \frac{3}{7} + \frac{2}{7}\frac{1}{2} + \frac{2}{7}\frac{1}{2}$ of a's in average, which cannot limit-surely classify between $\mathcal{A}_1, \mathcal{A}_2$.

From the point of view of practical applicability, limit-sure classifiers are the most powerful, although harder to study. In Section 4, we will study limit-sure classifiability, that we simply call classifiability. In Section 5, we further generalize this notion to a game-theoretic attack-classification framework, which is applicable in security settings.

3 Stationary distributions for HMMs

In order to solve limit-sure classification, we would like to use statistics on observations. Stationary distributions, which is a concept developed for *Markov chains*, tells us the frequency to expect about states, as used in [10]. We generalize this concept to HMMs to take into account observations. While stationary distributions for HMMs turn out to be crucial in the realm of classifiability, we believe it is also of independent interest.

For a Markov chain M, a stationary distribution σ is a distribution over states of M such that $\sigma \cdot M = \sigma$. In HMMs, the observation plays an important role and changes our knowledge of states in which the run could be. Thus, we consider the set of states that could be reached in an HMM $\mathcal{A}$ with a given observation, and call this as the *belief-state* or just *belief*. Formally, let w be an observation. The *belief* $B_{\mathcal{A}}(w)$ associated with w is the set of states $\{s^+(\rho) \mid obs(\rho) = w\}$ which can be reached by a path labeled by w. For instance, with the HMM $\mathcal{A}$ from Fig. 1, we have $B_{\mathcal{A}}(aa) = \{x, y\}$. We let $\mathcal{B}_{\mathcal{A}} = (2^S, \Delta, \mathbf{s_0})$ be the *belief automaton* associated with $\mathcal{A}$: (i) its states represent the beliefs associated with observations of $\mathcal{A}$, (ii) we have $(B, a, B') \in \Delta$ if $B' = \{s' \mid \exists s \in B, M(s, a, s) > 0\}$, and (iii) $\mathbf{s_0} = \{s \mid \sigma_0(s) > 0\} \in 2^S$. This is the usual subset construction used for determinizing an automaton, as shown on Fig. 1. As $\mathcal{B}_{\mathcal{A}}$ is deterministic, we sometimes abuse notation and denote $\Delta(B, a)$ for the unique B' with $(B, a, B') \in \Delta$.

Consider a BSCC D of HMM $\mathcal{A}$ (as for Markov chains, this is to ensure irreducibility). For $x \in D$, we denote by $\mathcal{B}^x_D$ the subgraph of $\mathcal{B}_{\mathcal{A}}$ reachable from $\{x\}$. On figure 1, we have $\mathcal{B}^y_D = \mathcal{B}_{\mathcal{A}}$. It has a unique BSCC, with 2 beliefs $\{x, y\}$ and $\{z\}$. We now show that this is the general form of the belief automaton:

▶ **Lemma 3.** *There is a unique BSCC in $\mathcal{B}^x_D$, and it does not depend upon $x \in D$.*

We denote E_D the set of beliefs X in the unique BSCC of $\mathcal{B}^x_D$, and $E_{\mathcal{A}}$ the union over all BSCCs D of $\mathcal{A}$. Notice that $E_{\mathcal{A}}$ may not contain all beliefs in the BSCCs of $\mathcal{B}_{\mathcal{A}}$, because we restrict ourselves to beliefs X reachable from $\{x\}$ with a single state x of a BSCC of $\mathcal{A}$. This is crucial for Lemma 3 to hold. We will see that considering singletons is not a

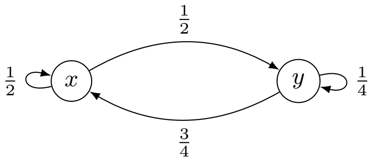


Figure 2 Markov chain $M_{x,y}$ associated with the belief $\{x, y\}$.

restriction: assume that the belief reached in a BSCC of beliefs comes from two different states x, y. Either the statistics on observation from x and y are the same, in which case we change nothing by considering them only from x. Otherwise, they have different statistics on observation, and looking at the observed statistics will give away with arbitrarily small error the state x or y which they originate from.

For Markov chains (i.e. HMMs on a one letter alphabet), the BSCC E_D is exactly $X_1 \to X_2 \cdots \to X_k \to X_1$, with k the *period* of this BSCC. Hence, this construction can be seen as a generalization to HMMs of the notion of period of a Markov chain. We use it to generalize the Fundamental theorem of Markov chains to HMMs.

Let $X \in E_{\mathcal{A}}$. We are interested in the *asymptotic distribution* associated to belief X, that is the statistics over states of X given that the belief state is X. From that, we will be able to deduce the statistics over observations. Let W_X the (possibly countable infinite) set of words which brings from belief X to belief X without seeing belief X in-between. Consider $\sigma_{y,i}$ the distribution over X such that $\sigma_{y,i}(x) = \sum_{w \in W_X^i} M(y, w, x)$, the probability of reaching x from y after seeing i words of W_X. To compute the limit of $\sigma_{y,i}$, we define the stationary distribution $\sigma_X : X \to [0, 1]$ of the HMM given a belief X. For that, we enrich the states of $\mathcal{A}$ with its beliefs, considering the product $\mathcal{A} \times \mathcal{B}_{\mathcal{A}}$ (same runs with same probabilities as in $\mathcal{A}$). For all $x, y \in X$, let $M_X(x, y)$ be the probability in the HMM $\mathcal{A} \times \mathcal{B}_{\mathcal{A}}$ to reach (y, X) from (x, X) before reaching any other (z, X), $z \neq y$ (we refer to [2] to compute $M_X(x, y)$ for all x, y). We have that for all $x \in X$, $\sum_{y \in X} M_X(x, y) = 1$, that is M_X is a Markov chain. For instance, on Fig. 1, let $X = \{x, y\} \in E$. The Markov chain M_X is depicted in Fig. 2 has a unique stationary distribution $\sigma(x) = \frac{3}{5}$ and $\sigma(y) = \frac{2}{5}$. We obtain:

▶ **Theorem 4.** *Given a HMM $\mathcal{A}$, let X be a belief in $E_{\mathcal{A}}$. Then, M_X has a unique stationary distribution denoted $\sigma_X : X \to [0, 1]$, i.e. $\sigma_X \cdot M_X = \sigma_X$. Further, for all $y \in X$, $\sigma_{y,i} \xrightarrow[i \to +\infty]{} \sigma_X$.*

Proof sketch. We apply the fundamental theorem to M_X to get the statement. It suffices to show that M_X is ergodic. For all $x \in X$, by Lemma 3, there is an observation v_x leading from $\{x\}$ to X, i.e. $\Delta(\{x\}, v_x) = X$. As $\Delta(\{x\}, v_x^i)$ is increasing with i and $|\Delta(\{x\}, v_x^i)| \leq n$ for all i, we obtain $\Delta(\{x\}, v_x^{n+1}) = \Delta(X, v_x^{n+1})$. We can then obtain a word w_x with $\Delta(\{x\}, w_x) = \Delta(X, w_x) = X$. Now, by induction on the size of X, we can build a uniform word w such that $\Delta(\{x\}, w) = X$ for all $x \in X$. For all $x, y \in X$, we get $M_X^{|w|}(x, y) > 0$. ◀

4 Limit-sure Classifiability

We start by stating the definition of *limit-sure classification* more precisely:

▶ **Definition 5.** *Two HMMs $\mathcal{A}_1, \mathcal{A}_2$ are limit-sure classifiable iff there exists a computable function, also called a classifier, $f : \Sigma^* \to \{1, 2\}$ such that $P(\rho$ run of $\mathcal{A}_1$ of size $k \mid f(obs(\rho)) = 2) \to_{k \to \infty} 0$, and similarly for ρ run of $\mathcal{A}_2$.*

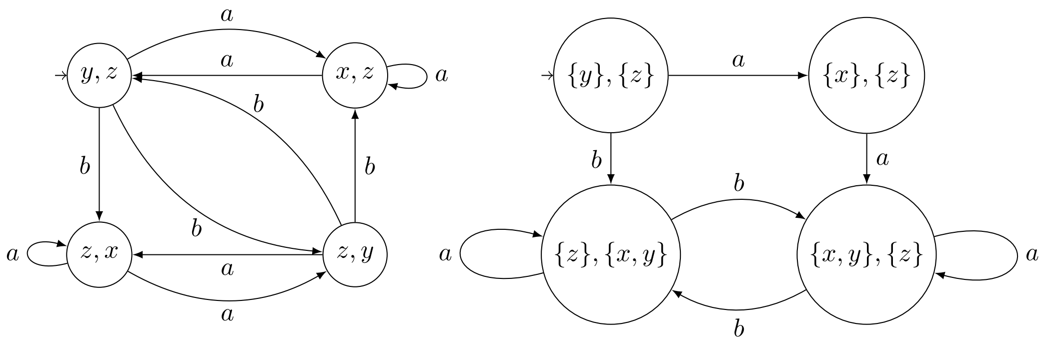


Figure 3 Twin automaton (on the left) and twin-belief automaton (on the right), for $\mathcal{A}_1, \mathcal{A}_2$ starting in states y and z.

(Notice we do not need $\perp$ as the classifier is allowed to give erroneous answers at first). Consider the *Maximum A Posteriori (MAP)* classifier [12, 10]: it answers 1 if $P^{\mathcal{A}_1}(u) > P^{\mathcal{A}_2}(u)$, and 2 otherwise. To do so, it just needs to record for every state of $\mathcal{A}_1$ (resp. every state of $\mathcal{A}_2$) the probability to observe u and finish in state s_1 (resp. s_2). Indeed, we may then compute confidence$(i, u) = \frac{P^{\mathcal{A}_i}(u)}{P^{\mathcal{A}_1}(u)+P^{\mathcal{A}_2}(u)}$, i.e. the probability that the decision i is correct after observing u. Notice that this confidence is not necessarily non-decreasing, and that the answer of a classifier can also switch from one answer to the other. In fact, we show in Proposition 16 in [1] that if $(\mathcal{A}_1, \mathcal{A}_2)$ is limit-sure classifiable, then the MAP classifier will be a limit-sure classifier. The main problem is to decide when limit-sure classification holds. In fact, this problem can be solved in PTIME. We remark that a variant of the problem was already shown to be in PTIME, namely distinguishability [6, 11]. While both problems coincide for HMMs, as explained in Section 4.4, our proof described in the rest of this section, crucially uses the notion of stationary distributions for HMMs developed in the previous section.

4.1 The Twin Automaton and the Twin Belief Automaton

Given HMMs $\mathcal{A}_1, \mathcal{A}_2$, we define their *twin automaton* $\mathcal{A} = (S = S_1 \times S_2, \Delta, s_0)$ as the product of the automata associated with $\mathcal{A}_1 \times \mathcal{A}_2$ by forgetting the probabilities. Recall that $\mathcal{A}_1$ has n states and $\mathcal{A}_2$ has m states. The transition relation is $\Delta = \{((s_1, s_2), a, (t_1, t_2)) \mid M_{\mathcal{A}_1}(s_1, a, t_1) > 0, M_{\mathcal{A}_2}(s_2, a, t_2) > 0\}$, with initial state $s_0 = (s_0^1, s_0^2)$. We call states of $\mathcal{A}$ *twin states*. In the following, we will often consider the belief automata $\mathcal{B}_{\mathcal{A}}, \mathcal{B}_{\mathcal{A}_1}, \mathcal{B}_{\mathcal{A}_2}$ associated with $\mathcal{A}, \mathcal{A}_1, \mathcal{A}_2$, obtained by the subset construction (see Section 3). States of $\mathcal{B}_{\mathcal{A}}$ will be called *twin beliefs*. Notice that although twin beliefs are formally sets of pairs of states in $2^{S_1 \times S_2}$, we can also present them as pairs of sets of states $2^{S_1} \times 2^{S_2}$ because if (s_1, s_2) and (s_1', s_2') are in the same twin belief, then we also have (s_1, s_2') and (s_1', s_2) in this twin belief. We will thus write the twin belief $X(u)$ associated with observation u as $X(u) = (X_1(u), X_2(u))$, with $X_1(u), X_2(u)$ the beliefs states of $\mathcal{B}_{\mathcal{A}_1}, \mathcal{B}_{\mathcal{A}_2}$ associated with u. Figure 3 presents an example with a twin automaton and the twin belief automaton for two copies of the HMM given in figure 1, one starting in state y and the other starting in state z.

▶ **Lemma 6** (Proposition 18 in [6]). *Let (X_1', X_2') be a reachable twin belief of $\mathcal{B}_{\mathcal{A}}$. Let $X_1 \subseteq X_1', X_2 \subseteq X_2'$. Let σ_1, σ_2 be two distributions over X_1, X_2 with $(\mathcal{A}_1, \sigma_1) \equiv (\mathcal{A}_2, \sigma_2)$. Then one cannot classify between $\mathcal{A}_1, \mathcal{A}_2$.*

4.2 Characterization for classifiability

Our goal is to use the result of Section 3 to obtain stationary distributions in $\mathcal{A}_1, \mathcal{A}_2$, and classify between them by comparing the stochastic language wrt these stationary distributions using probabilistic equivalence (see Section 2.1). In order to do this, we first need to compare the same information in both HMMs. The idea is to consider twin beliefs from each HMM: we will enrich $\mathcal{A}_1$ with the beliefs of $\mathcal{A}_2$, and vice versa. Let $\mathcal{A}'_1$ be the HMM where the state space is $S_1 \times 2^{S_2}$, and the transition matrix is $M_{\mathcal{A}'_1}((x,Y),a,(x',Y')) = M_{\mathcal{A}_1}(x,a,x')$ if $Y' = \{y' \mid (y,a,y'), y \in Y\}$, and 0 otherwise, for all x, Y, a, x', Y'. We define similarly $\mathcal{A}'_2$ with set of states $S_2 \times 2^{S_1}$. It is easy to see that for all observation w, the belief state $B_{\mathcal{A}'_1}(w) = \{(x_1, B_{\mathcal{A}_2}(w)) \mid x_1 \in B_{\mathcal{A}_1}(w)\}$, is isomorphic to the twin belief $(B_{\mathcal{A}_1}(w), B_{\mathcal{A}_2}(w))$, isomorphic to $B_{\mathcal{A}'_2}(w)$, and we will abuse notation and represent beliefs of $\mathcal{A}'_1$ and $\mathcal{A}'_2$ as twin belief (X_1, X_2), where X_1 or X_2 can be empty.

What we are interested in is what happens after a BSCC of $\mathcal{A}$ is reached. We thus consider twin beliefs reachable from some (x_1, x_2) in the BSCC of $\mathcal{A}$. The set of twin beliefs reachable in $\mathcal{A}'_1$ and in $\mathcal{A}'_2$ from $(\{x_1\},\{x_2\})$ are almost the same, except for twin beliefs of the form $(X_1, \emptyset)$ which cannot be reached in $\mathcal{A}'_2$, and of the form $(\emptyset, X_2)$ which cannot be reached in $\mathcal{A}'_1$.

▶ **Definition 7.** *We say that a twin belief (X_1, X_2) is* oblivious *if the languages of $\mathcal{B}_{\mathcal{A}_1}$ from X_1 and of $\mathcal{B}_{\mathcal{A}_2}$ from X_2 are the same.*

By definition, if (X_1, X_2) is not oblivious, there are words differentiating X_1 and X_2.

Now, assume that $X = (X_1, X_2)$ is oblivious. The twin beliefs reachable from (X_1, X_2) are the same in $\mathcal{A}'_1$ and $\mathcal{A}'_2$. To potentially differentiate them, we need to consider their long term statistics. Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be the belief automata associated with $\mathcal{A}'_1$ and $\mathcal{A}'_2$. Let $E_{\mathcal{A}}$ be the union of BSCCs of twin beliefs accessible from twin states in the BSCCs of twin states, as in lemma 3. Let $X \in E_{\mathcal{A}}$. In this case, we say that X is in the BSCCs of twin beliefs. We define $\sigma^1_X : X_1 \to [0,1]$ the *stationary distribution* in $\mathcal{A}'_1$ around the twin belief X (formally, σ^1_X is defined on (x, X_2) for all $x \in X_1$, and we omit the second component X_2 because it is constant). In the same way, we define $\sigma^2_X : X_2 \to [0,1]$ for the second component X_2 around the twin belief X. We can then look for words differentiating $\mathcal{A}_1, \mathcal{A}_2$, i.e. with different probabilities from σ^1_X and from σ^2_X. We can now state our characterization:

▶ **Theorem 8.** *The following are equivalent:*

1. *One cannot limit-surely classify between $\mathcal{A}_1, \mathcal{A}_2$,*
2. *There exists an oblivious $X \in E_{\mathcal{A}}$ in a BSCC of twin beliefs such that $(\mathcal{A}_1, \sigma^1_X) \equiv (\mathcal{A}_2, \sigma^2_X)$,*
3. *There exists a BSCC D of $\mathcal{A}$ and $X_1 \subseteq S_1, X_2 \subseteq S_2$, and $y_1 \in X_1$, $y_2 \in X_2$, such that $(y_1, x_2) \in D$ for all $x_2 \in X_2$ and $(x_1, y_2) \in D$ for all $x_1 \in X_1$, and two distributions σ^1 over X_1 and σ^2 over X_2 such that $(\mathcal{A}_1, \sigma^1) \equiv (\mathcal{A}_2, \sigma^2)$.*

The second condition is sufficient to show that MAP is a limit-sure classifier (see Proposition 16 in [1]). However, checking condition 2 explicitly is not algorithmically efficient, as the belief automaton can have exponentially many states. Instead, to obtain a PTIME algorithm to check limit-sure classifiability, we will use the third condition. For comparison, in [6], a variant of the equivalence between (1) and (3) is shown, without using the stationary distributions σ^1_X, σ^2_X of (2).

For the proof, we note that the case of 2 implies 3 is easy. For the remaining two directions, i.e. 1 implies 2 and 3 implies 1, proofs are technical, and can be found in the long version [1]. For 1 implies 2, we prove that negation of 2 implies that the MAP classifier (defined in beginning of Section 4) is limit-sure, implying negation of 1. Intuitively, negation of 2 means that every pair of reachable beliefs have a distinguishing word. It then suffices to consider statistics on these finite number of distinguishing words to know the originating HMM with arbitrarily high probability. For 3 implies 1, we show that any twin belief (H_1, H_2) reached from (y_1, y_2) in $E_{\mathcal{A}}$ must be oblivious because of the probabilistic equivalence. We show this implies $(\mathcal{A}_1, \sigma^1_{H_1,H_2})$ and $(\mathcal{A}_2, \sigma^2_{H_1,H_2})$ are equivalent and conclude using Lemma 6.

4.3 A PTIME Algorithm

Theorem 8 gives us a characterization for the existence of a limit-sure classifier. The third condition is particularly interesting, because it does not require computing beliefs. Using this, we can build an efficient algorithm, similar to [6], to test in PTIME whether there exists a limit-sure classifier between $\mathcal{A}_1, \mathcal{A}_2$.

Our Algorithm 1, presented below, uses linear programming: we let $v_1, \ldots, v_\ell$ be the basis of $Eq(\mathcal{A}_1, \mathcal{A}_2)$ (see Section 2.1). There exist two distributions σ^1, σ^2 over X_1, X_2 with $(\mathcal{A}_1, \sigma^1) \equiv (\mathcal{A}_2, \sigma^2)$ iff the linear system of equations (for all $j \leq \ell$, $(\sigma^1 \quad \sigma^2) \cdot v_j = 0$) has a solution (with σ^1, σ^2 as variables), which can be solved in Polynomial time.

Algorithm 1 Limit-sure Classifiability.

```
Compute $D_1, \ldots, D_k$ the BSCCs of the twin automaton $\mathcal{A}$.
for i=1..k do
    for $(y_1, y_2) \in D_i$ do
        Let $X_1 = \{x_1 \mid (x_1, y_2) \in D_i\}$, $X_2 = \{x_2 \mid (y_1, x_2) \in D_i\}$.
        if there exist two distributions $\sigma^1, \sigma^2$ over $X_1, X_2$ with $\sigma^1(y_1) > 0$ and $\sigma^2(y_2) > 0$
            with $(\mathcal{A}_1, \sigma^1) \equiv (\mathcal{A}_2, \sigma^2)$ then
            return not classifiable
return classifiable
```

The correctness of the algorithm is immediate from Theorem 8, as it checks explicitly for the third condition to hold, in which case it returns not classifiable. If the third condition is false for every BSCC D, then it returns classifiable.

4.4 Comparison with Distinguishability between HMMs [11]

We complete this section, by comparing our results with a related result on HMMs. In [11], the problem of distinguishability between labeled Markov Chains has been considered. First, labeled Markov Chains are just another name for HMMs. The idea behind distinguishability is similar to the idea behind classifiability. Still, there are some technical differences: distinguishability asks that for all $\varepsilon > 0$, there exists a $(1-\varepsilon)$-*classifier*, that is a classifier $f : \Sigma^* \to \{\bot, 1, 2\}$, such that if the classifier answers $f(u) = 1$, then there is probability at least $(1-\varepsilon)$ that the observation comes from a run from $\mathcal{A}_1$, and similarly for $f(u) = 2$. To compare, limit-sure classifiers need to be uniform over ε (see the next section).

The authors of [11] show that this notion can be checked in PTIME, by indirectly using the result of [6] stating that one can check in PTIME whether the total variation distance between two HMMs is 1. More precisely, the total variation distance is defined as:

▶ **Definition 9.** *The* total variation distance *between two HMMs $\mathcal{A}_1$ and $\mathcal{A}_2$ is given by*

$$d(\mathcal{A}_1, \mathcal{A}_2) = \sup_{E \subset \Sigma^\omega} |P_{\mathcal{A}_1}(E) - P_{\mathcal{A}_2}(E)|.$$

This supremum has been shown to be a maximum [6]. It is not too hard to show that limit-sure classification coincides with these notions as well for HMMs:

▶ **Theorem 10.** *The following are equivalent:*

1. *There exists a limit-sure classifier for* $\mathcal{A}_1, \mathcal{A}_2$*,*
2. *For all* $\varepsilon > 0$*, there exists a* $(1-\varepsilon)$-classifier *for* $\mathcal{A}_1, \mathcal{A}_2$*,*
3. $d(\mathcal{A}_1, \mathcal{A}_2) = 1$*.*

The proofs to obtain the PTIME algorithms are quite different though: we use stationary distributions in HMMs while [6] focuses on separating events. Some intermediate results are however related: our Proposition 18 in [1] is to be compared with Proposition 19 b) of [6]: Our statement is stronger as the equivalence is true from *all* pairs of states with the same (non stochastic) language - and in particular from $(i_1, j_1) = (y_1, y_2)$ (cf Proposition 17 in [1]). Also, the proof of Proposition 18 in [1] is simple, using strict convexity focusing on *one* finite separating word, while in [6], the existence of a maximal separating events (sets of infinite words) is used crucially in the proof of Proposition 19 b).

Surprisingly, our resulting algorithm is very similar to the one in [6], whereas we use very different methods. Still, we can restrict the search to *distributions* in a BSCC of twin states, while [6] considers *subdistributions* on the whole state space of twin states. This allows us to optimize the number of variables in the Linear Program.

5 Attack-classification

While limit-sure classification allows for some misclassification, i.e. error in classification, it requires that every execution of the HMMs is classifiable. From a security perspective, if one wants to make sure that two systems cannot be distinguished from each other, then the question changes slightly: from the point of view of an attacker who could exploit the knowledge of which model the system is following, it need not classify every single execution. It only needs to find one execution for which it can decide. This gives rise to what we call *attack-classification*, which amounts to providing the attacker with a reset action she can play when she believes the execution cannot be classified. Then, a new (possibly the same) HMM is taken at random and an execution of this new HMM is observed by the attacker. For instance, it is not possible to limit-surely classify between HMM $\mathcal{A}_3$ and HMM $\mathcal{A}_4$ on Figure 4, because executions starting with a b cannot be classified. On the other hand, an attacker can wait for an execution of the system starting with an a, for which he is sure the HMM is $\mathcal{A}_3$. If it starts with a b, then the attacker just forgets this execution and wait for a new execution of the system (the "reset" operation).

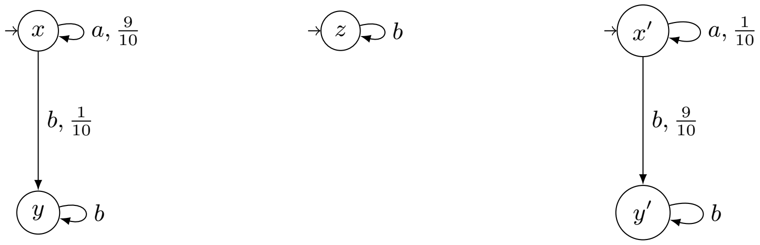


Figure 4 HMMs $\mathcal{A}_3, \mathcal{A}_4$ and $\mathcal{A}_5$ (left to right). One cannot classify betweeen $\mathcal{A}_3, \mathcal{A}_4$, but they can be attack-classified. On the other hand, one cannot attack-classify between $\mathcal{A}_3, \mathcal{A}_5$.

We start by considering *limit-sure attack-classifiers*, namely, we require that there exists a *reset-strategy*, which with probability 1, resets only finitely many times, and a limit-sure classifier for the observation after the last reset. We also consider what happens if instead of limit-sure classifier, we ask for the existence of a family of $(1-\varepsilon)$-classifiers after the last reset, one for each ε. The difference is that the reset action can take into account the ε in the latter, but not in the former. While both notions coincide for the classifiers defined in the previous section, we show now that they do not coincide for attack-classification.

Figure 4 illustrates the difference between these two notions, considering $\mathcal{A}_3$ and $\mathcal{A}_5$. First, for all $\varepsilon > 0$, there exists an $(1-\varepsilon)$-attack-classifier: given an ε, the reset strategy resets if the first letter b happens within the first $k_\varepsilon = log(\frac{1}{9\varepsilon})$ steps. That is, the reset strategy is $\tau(a^*) = \bot$, $\tau(a^{k_\varepsilon}w) = \bot$ and $\tau(a^\ell b) = reset$ for $\ell < k_\varepsilon$. For observation $a^{k_\varepsilon}w$, the classifier claims that the HMM is $\mathcal{A}_3$, which is true with probability at least $(1-\varepsilon)$. However, this reset strategy is not compatible with limit-sure classifier (and, in fact, no reset strategy is), because it is not uniform wrt all ε: once a b has been produced, no more information can be gathered. On the other hand, limit-sure attack-classified implies the existence of $(1-\varepsilon)$-attack-classifiers for all ε. Thus the former notion of limit-sure attack-classifier is strictly contained in the latter. More importantly, we show that deciding the former is PSPACE-complete, while the latter turns out to be undecidable.

5.1 Limit-sure attack-classifiability is PSPACE-complete

Let us first formalize the definition of attack-classification.

▶ **Definition 11.** *We say two HMMs $\mathcal{A}_1, \mathcal{A}_2$ are* limit-sure attack-classifiable *if: there exists*

1. reset strategy $\tau : \Sigma^* \to \{\bot, reset\}$ *telling when to reset, and which eventually stops resetting, with probability 1 on the reset runs, and*
2. *limit-sure classifier for u, where $u \in \Sigma^*$ denotes the suffix of observations since last reset.*

In the following, we show an algorithmic characterization for this concept. Intuitively, there needs to exist one execution of one HMM (say $\mathcal{A}_1$), such that no matter the execution of the other HMM with the same observation, we can eventually classify between these two executions. We will thus consider $\mathcal{A}'_1$ and $\mathcal{A}'_2$, the HMMs $\mathcal{A}_1$ and $\mathcal{A}_2$ enriched with the beliefs of the other HMM.

First, we define classifiable twin states in the BSCC of twin states: $(x_1, x_2) \in \mathcal{A}$ is classifiable iff for (X_1, X_2) in the unique BSCC of twin beliefs, either (X_1, X_2) is non oblivious or (X_1, X_2) is oblivious and $(\mathcal{A}_1, \sigma^1_{X_1,X_2}) \not\equiv (\mathcal{A}_2, \sigma^2_{X_1,X_2})$, for $(\sigma^1_{X_1,X_2}, \sigma^2_{X_1,X_2})$ the stationary distributions built for (X_1, X_2). Notice that it does not depends upon the choice of (X_1, X_2). For a belief state X_2 of $\mathcal{A}_2$, we say that $(x_1, X_2) \in \mathcal{A}'_1$ is classifiable if (x_1, x_2) is classifiable for all $x_2 \in X_2$ (in particular, every (x_1, x_2) is in a BSCC of twin states). In particular, $(x_1, \emptyset)$ is classifiable. We define $(x_2, X_1) \in \mathcal{A}'_2$ similarly.

▶ **Proposition 12.** $(\mathcal{A}_1, \mathcal{A}_2)$ *is limit-sure attack-classifiable iff there exists a classifiable $(x_1, X_2) \in \mathcal{A}'_1$, or a classifiable $(x_2, X_1) \in \mathcal{A}'_2$.*

In case there are more than two HMMs, we follow the state s of one HMM and the belief of every other HMMs along the observation, and we need to check classifiability between (s, t) for every t in the belief of any of the other HMMs. Using this characterization, we obtain:

▶ **Theorem 13.** *Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two HMMs. It is PSPACE-complete to check whether $(\mathcal{A}_1, \mathcal{A}_2)$ are limit-sure attack-classifiable.*

5.2 Existence of $(1 - \varepsilon)$ attack-classifiers for all ε is undecidable

We now turn to the other notion. Let $\varepsilon > 0$. An $(1 - \varepsilon)$ attack-classifier for two HMMs $\mathcal{A}_1, \mathcal{A}_2$ is given by:

1. A reset strategy $\tau : \Sigma^* \to \{\perp, reset\}$ telling when to reset, and which eventually stops resetting, with probability 1 on the reset runs, and
2. a $(1 - \varepsilon)$-classifier for u, where $u \in \Sigma^*$ denotes the suffix of the observations since the last reset.

We next show that this notion, which we showed to be weaker than limit-sure attack-classifiability on Fig 4, is also computationally much harder, in fact, it is undecidable.

▶ **Theorem 14.** *It is undecidable to know whether for all ε, there exists an $(1 - \varepsilon)$ attack-classifier between 2 HMMs.*

Intuitively, we reduce from the problem of whether a PFA $\mathcal{B}$, that accepts all words with probability in $(0, 1)$, is 0 and 1 isolated, that is, there is no sequence of words $(w_i)_{i \in \mathbb{N}}$ such that $lim_{n \to \infty} P^{\mathcal{B}}(w_i) = 0$ or $= 1$. This problem is undecidable [8]. The idea is to transform the PFA into an HMM which performs the actions of the PFA uniformly at random. We check whether we can attack classify this HMM with an HMM which accepts all words of size k with probability $1/2^k$. This is possible if 0 is not isolated or if 1 is not isolated.

6 Conclusion

In this paper, we tackled the notion of limit-sure classifiability between HMMs, which is a general notion in studying how to uncover hidden information in partially observable systems. The class of classifiers we consider are quite powerful, as they can use statistics on the observations in order to take their decision. To obtain our results, summarized in the table below we developed a robust theory of stationary distributions for HMMs.

While limit-sure classifiability is stronger and more complex than almost-sure classifiability, checking for it is in a lower complexity class: PTIME instead of PSPACE-complete. This result shines some new light on total variation metric for stochastic systems, recovering with different techniques the PTIME result from [6]. We also considered attack-classifiability, where the attacker needs to classify at least one observation rather than every execution. In this setting, there is a difference between limit-sure classifier and the existence of $(1-\varepsilon)$-classifiers for each ε. Limit-sure attack-classifiability is decidable (PSPACE-complete), whereas the existence of $(1 - \varepsilon)$-classifiers for all ε is undecidable.

	limit-sure classifiability	limit-sure attack-classifiability	$\forall\varepsilon, (1-\varepsilon)$ attack-classifiability
Complexity	PTIME	PSPACE-complete	Undecidable

References

1 S. Akshay, Hugo Bazille, Eric Fabre, and Blaise Genest. Classification among Hidden Markov Models, 2019. URL: http://perso.crans.org/genest/ABFG19.pdf.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, Cambridge, MA, 2008.

3 Vijay Balasubramanian. Equivalence and Reduction of Hidden Markov Models, 1993.

4 Nathalie Bertrand, Serge Haddad, and Engel Lefaucheux. Foundation of diagnosis and predictability in probabilistic systems. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science*, volume 29 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 417–429. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2014.

5 Nathalie Bertrand, Serge Haddad, and Engel Lefaucheux. Accurate approximate diagnosability of stochastic systems. In *Language and automata theory and applications*, volume 9618 of *Lecture Notes in Comput. Sci.*, pages 549–561. Springer, 2016. doi:10.1007/978-3-319-30000-9_42.

6 Taolue Chen and Stefan Kiefer. On the total variation distance of labelled Markov chains. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages Article No. 33, 10. ACM, New York, 2014.

7 Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Equivalence of labeled Markov chains. *Internat. J. Found. Comput. Sci.*, 19(3):549–563, 2008. doi:10.1142/S0129054108005814.

8 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: decidable and undecidable problems. In *Automata, languages and programming. Part II*, volume 6199 of *Lecture Notes in Comput. Sci.*, pages 527–538. Springer, Berlin, 2010. doi:10.1007/978-3-642-14162-1_44.

9 John G. Kemeny and J. Laurie Snell. *Finite Markov chains.* The University Series in Undergraduate Mathematics. D. Van Nostrand Co., Inc., Princeton, N.J.-Toronto-London-New York, 1960.

10 Christoforos Keroglou and Christoforos N. Hadjicostis. Probabilistic system opacity in discrete event systems. *Discrete Event Dyn. Syst.*, 28(2):289–314, 2018. doi:10.1007/s10626-017-0263-8.

11 Stefan Kiefer and A. Prasad Sistla. Distinguishing hidden Markov chains. In *Proceedings of the 31st Annual ACM-IEEE Symposium on Logic in Computer Science (LICS 2016)*, page 10. ACM, New York, 2016.

12 Daniel Ramage. Hidden Markov Models Fundamentals, CS229, lecture notes, 2007.

13 Anooshiravan Saboori and Christoforos N. Hadjicostis. Probabilistic current-state opacity is undecidable. In *Proceedings of the 19th Intl. Symposium on Mathematical Theory of Networks and Systems*, pages 1–10, 2010.

14 Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. Contr. Sys. Techn.*, 4(2):105–124, 1996. doi:10.1109/87.486338.

15 David Thorsley and Demosthenis Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Trans. Automat. Control*, 50(4):476–492, 2005. doi:10.1109/TAC.2005.844722.

16 Wen-Guey Tzeng. The Equivalence and Learning of Probabilistic Automata (Extended Abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989, IEEE FOCS'89*, pages 268–273, 1989. doi:10.1109/SFCS.1989.63489.

Minimisation of Event Structures

Paolo Baldan
University of Padova, Italy
baldan@math.unipd.it

Alessandra Raffaetà
Ca' Foscari University of Venice, Italy
raffaeta@unive.it

Abstract

Event structures are fundamental models in concurrency theory, providing a representation of events in computation and of their relations, notably concurrency, conflict and causality. In this paper we present a theory of minimisation for event structures. Working in a class of event structures that generalises many stable event structure models in the literature, (e.g., prime, asymmetric, flow and bundle event structures) we study a notion of behaviour-preserving quotient, taking hereditary history preserving bisimilarity as a reference behavioural equivalence. We show that for any event structure a uniquely determined minimal quotient always exists. We observe that each event structure can be seen as the quotient of a prime event structure, and that quotients of general event structures arise from quotients of (suitably defined) corresponding prime event structures. This gives a special relevance to quotients in the class of prime event structures, which are then studied in detail, providing a characterisation and showing that also prime event structures always admit a unique minimal quotient.

2012 ACM Subject Classification Theory of computation → Concurrency; Software and its engineering → Formal methods

Keywords and phrases Event structures, minimisation, history-preserving bisimilarity, behaviour-preserving quotient

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.30

Related Version Extended version of the paper available at `http://arxiv.org/abs/1907.07042`.

Funding *Paolo Baldan*: Supported by the PRIN Project Analysis of Program Analyses (ASPRA).

1 Introduction

When dealing with formal models of computational systems, a classical problem is that of minimisation, i.e., for a given system, define and possibly construct a compact version of the system which, very roughly speaking, exhibits the same behaviour as the original one, avoiding unnecessary duplications. The minimisation procedure depends on the notion of behaviour of interest and also on the expressive power of the formalism at hand, which determines its capability of describing succinctly some behaviour. A classical example is that of finite state automata, where one is typically interested in the accepted language. Given a deterministic finite state automaton, a uniquely determined minimal automaton accepting the same language can be constructed, e.g., as a quotient of the original automaton via a partition/refinement algorithm (see, e.g., [16]). Moving to non-deterministic finite automata, minimal automata become smaller, at the price of a computationally more expensive minimisation procedure and non-uniqueness of the minimal automaton [22].

In this paper we study the problem of minimisation for event structures, a fundamental model in concurrency theory [33, 34]. Event structures are a natural semantic model when one is interested in modelling the dynamics of a system by providing an explicit representation of the events in computations (occurrence of atomic actions) and of the relations between

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 30; pp. 30:1–30:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

events, like causal dependencies, choices, possibility of parallel execution, i.e., in what is referred to as a true concurrent (non-interleaving) semantics. Prime event structures [24], probably the most widely used event structure model, capture dependencies between events in terms of causality and conflict. A number of generalisations of prime event structures have been introduced in the literature. For instance, flow [9, 8] and bundle [20] event structures add the possibility of directly modelling disjunctive causes. Asymmetric event structures [5] and extended bundle event structures [20] include an asymmetric form of conflict which allows one to model concurrent readings and precedences between actions. Event structures have been used for defining a concurrent semantics of several formalisms, like Petri nets [24], graph rewriting systems [4, 3, 27] and process calculi [32, 31, 10]. Recent applications are in the field of weak memory models [11, 25, 17, 13] and of process mining and differencing [15].

Behavioural equivalences, defined in a true concurrent setting, take into account not only the possibility of performing steps, but also the way in which such steps relate with each other. We will focus on hereditary history preserving (hhp-)bisimilarity [7], the finest equivalence in the true concurrent spectrum in [29], which, via the concept of open map, has been shown to arise as a canonical behavioural equivalence when considering partially ordered computations as observations [18].

The motivation for the present paper originally stems from some work on business process models. The idea, advocated in [15, 1], is to use event structures as a foundation for representing, analysing and comparing process models. Processes are mined in the form of event structures and then translated into a suitable process modeling language. The processes, in their graphical presentation, should be simple and understandable, as much as possible, by a human user, who should be able, e.g., to interpret the differences between two processes diagnosed by a comparison tool. For this aim it is important to avoid "redundancies" in the representation and thus to reduce the number of events, without altering the behaviour (modifications that "extend" the behaviour could be sensible in model generalisation, but this is not of interest here). The paper [2] explores the use of asymmetric and flow event structures and, for such models, it introduces some reduction techniques that allow one to merge events without changing the true concurrent behaviour. A notion of behaviour-preserving quotient, referred to as a folding, is introduced over an abstract class of event structures, having asymmetric and flow event structures as subclasses. However, no general theory is developed. The mentioned paper focuses on a special class of foldings, the so-called elementary foldings, which can only merge a single set of events into one event, and these are studied separately on each specific subclass of event structures (asymmetric and flow event structures), providing only sufficient conditions ensuring that a function is an elementary folding.

A general theory of behaviour-preserving quotients for event structures is thus called for, settling some natural foundational questions. Is the notion of folding adequate, i.e., are all behaviour-preserving quotients expressible in terms of foldings? Is there a minimal quotient in some suitably defined general class of event structures? Does it exist in specific subclasses? (for asymmetric and flow event structures the answer is known to be negative, but for prime event structures the question is open). Can we have a characterisation of foldings for specific subclasses of event structures, providing not only sufficient but also necessary conditions?

In this paper we start addressing the above questions. We work in a general class of event structures based on the idea of family of posets [26] (also called rigid families in [14]). Although this is not discussed in deep due to space limitations, poset event structures are sufficiently expressive to generalise most stable event structures models in the literature, including prime [24], asymmetric [5], flow [9] and bundle [20] event structures (a wider discussion can be found in [6].)

As a first step we study, in this general setting, the notion of folding. A folding is a surjective function that identifies some events while keeping the behaviour unchanged. Formally, it establishes a hhp-bisimilarity between the source and target event structure. It turns out that not all behaviour-preserving quotients arise as foldings, but we show that for any behaviour-preserving quotient, there is a folding that induces a coarser equivalence. Additionally, given two possible foldings of an event structure we show that it is always possible to "join" them. This allows us to prove that for each event structure a maximally folded version, namely a uniquely determined minimal quotient, always exists.

Relying on the order-theoretic properties of the set of configurations of event structures [26], and on the correspondence between prime event structures and domains [24], we derive that each event structure in the considered class arises as the folding of a canonical prime event structure. Moreover, all foldings between general event structures arise from foldings of the corresponding canonical prime event structures. Interestingly, this result can be derived from a characterisation of foldings as open maps in the sense of [18].

The results above give a special relevance to foldings in the class of prime event structures, which thus are studied in detail. We provide necessary and sufficient conditions characterising foldings for prime event structures. This allows us to establish a clear connection with the so-called abstraction morphisms, introduced in [12] for similar purposes. The characterisation of foldings provided can guide, at least in the case of finite structures, the effective construction of behaviour preserving quotients. Moreover we show that also prime event structures always admit a minimal quotient.

Most results have a natural categorical interpretation, which is only hinted at in the paper. In order to keep the presentation simple, the categorical references are inserted in side remarks that can be safely skipped by the non-interested reader. This applies, in particular, to the possibility of viewing foldings as open maps in the sense of [18]. This correspondence, which in the present paper only surfaces, suggests the possibility of understanding and developing our results in a more abstract categorical setting. More details are provided in the extended version [6].

The rest of the paper is structured as follows. In § 2 we introduce the class of event structures we work with and hereditary history preserving bisimilarity. Moreover, we discuss how some event structure models in the literature embed into the considered class. In § 3 we study the notion of folding, we prove the existence of a minimal quotient and we show the tight relation between general foldings and those on prime event structures. In § 4 we present folding criteria on prime event structures, and discuss the existence of minimal quotients. Finally, in § 5 we draw some conclusions, discuss connections with related literature and outline future work venues. Due to space limitations all proofs have been omitted. They can be found in the extended version [6], which also contains some additional results.

2 Event Structures and History Preserving Bisimilarity

In this section we define *hereditary history-preserving bisimilarity*, the reference behavioural equivalence in the paper. This is done for an abstract notion of event structure, introduced in [26], in a way that various stable event structure models in the literature can be seen as special subclasses. We will explicitly discuss prime [24] and flow [9, 8] event structures.

Notation. We first fix some basic notation on sets, relations and functions. Let $r \subseteq X \times X$ be a binary relation. The relation r is *acyclic* on Y if there is no $\{y_0, y_1, \ldots, y_n\} \subseteq Y$ such that $y_0 \; r \; y_1 \; r \ldots r \; y_n \; r \; y_0$. Relation r is a *partial order* if it is reflexive, antisymmetric and

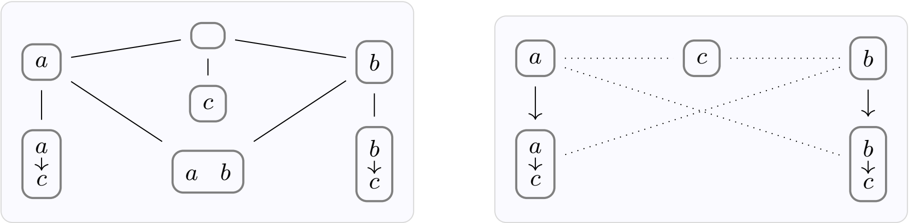


Figure 1 An event structure E and the canonical PES $\mathbb{P}(\mathsf{E})$.

transitive. Given a function $f : X \to Y$ we will denote by $f[x \mapsto y] : X \cup \{x\} \to Y \cup \{y\}$ the function defined by $f[x \mapsto y](x) = y$ and $f[x \mapsto y](z) = f(z)$ for $z \in X \setminus \{x\}$. Note that the same notation can represent an update of f, when $x \in X$, or an extension of its domain, otherwise. For $Z \subseteq X$, we denote by $f_{|Z} : Z \to Y$ the restriction of f to Z.

2.1 Event Structures

Following [26, 28, 30, 2, 14], we work on a class of event structures where configurations are given as a primitive notion. More precisely, we borrow the idea of family of posets from [26], more recently considered also under the name of rigid family in [14].

▶ **Definition 2.1** (family of posets). *A* poset *is a pair $(C, \leq_C)$ where C is a set and $\leq_C$ is a partial order on C. A poset will be often denoted simply as C, leaving the partial order relation $\leq_C$ implicit. Given two posets C_1 and C_2 we say that C_1 is a* prefix *of C_2 and write $C_1 \sqsubseteq C_2$ if $C_1 \subseteq C_2$ and $\leq_{C_1} = \leq_{C_2} \cap (C_2 \times C_1)$. A* family of posets *$F$ is a prefix-closed set of finite posets i.e., a set of finite posets such that if $C_2 \in F$ and $C_1 \sqsubseteq C_2$ then $C_1 \in F$. We say that two posets $C_1, C_2 \in F$ are compatible, written $C_1 \frown C_2$, if they have an upper bound, i.e., there is $C \in F$ such that $C_1, C_2 \sqsubseteq C$. The family of posets F is called* coherent *if each subset of F whose elements are pairwise compatible has an upper bound.*

Posets C will be used to represent configurations, i.e., sets of events executed in a computation of an event structure. The order $\leq_C$ intuitively represents the order in which the events in C can occur. This motivates the prefix order that can be read as a computational extension: when $C_1 \sqsubseteq C_2$ we have that $C_1 \subseteq C_2$, events in C_1 are ordered exactly as in C_2, and the new events in $C_2 \setminus C_1$ cannot precede events already in C_1 (i.e., for all $x_1 \in C_1$, $x_2 \in C_2$, if $x_2 \leq_{C_2} x_1$ then $x_2 \in C_1$).

An example of family of posets can be found in Fig. 1 (left). Observe, for instance, that the configuration with set of events $\{c\}$ is not a prefix of the one with set of events $\{a, c\}$, since in the latter $a \leq c$.

An event structure is then defined simply as a coherent family of posets where events carry a label. Hereafter Λ denotes a fixed set of labels.

▶ **Definition 2.2** ((poset) event structure). *A* (poset) event structure *is a tuple $\mathsf{E} = \langle E, \mathit{Conf}(\mathsf{E}), \lambda\rangle$ where E is a set of events, $\mathit{Conf}(\mathsf{E})$ is a coherent family of posets such that $E = \bigcup \mathit{Conf}(\mathsf{E})$ and $\lambda : E \to \Lambda$ is a labelling function. For a configuration $C \in \mathit{Conf}(\mathsf{E})$ the order $\leq_C$ is referred to as the* local order.

In [2] abstract event structures are defined as a collection of ordered configurations, without any further constraint. This is sufficient for giving some general definitions which are then studied in specific subclasses of event structures. Here, in order to develop a theory of foldings at the level of general event structures, we need to assume stronger properties, those of a family of posets from [26] (e.g, the fact that Definition 3.14 is well-given relies on this). This motivates the name poset event structure. Also note that, differently from what happens in other general concurrency models, like configuration structures [30], configurations are endowed explicitly with a partial order, which in turn intervenes in the definition of the prefix order between configurations. This is essential to view as subclasses some kinds of event structures, like asymmetric event structures [5] or extended bundle event structures [21], where the order on configuration is not simply subset inclusion. Since we only deal with poset event structures and their subclasses, we will often omit the qualification "poset" and refer to them just as event structures. Moreover, we will often identify an event structure E with the underlying set E of events and write, e.g., $x \in \mathsf{E}$ for $x \in E$.

An *isomorphism of configurations* $f : C \to C'$ is an isomorphism of posets that respects the labelling, namely for all $x, y \in C$, we have $\lambda(x) = \lambda(f(x))$ and $x \leq_C y$ iff $f(x) \leq_{C'} f(y)$. When configurations C, C' are isomorphic we write $C \simeq C'$.

Given an event x in a configuration C, it will be useful to refer to the prefix of C including only those events that necessarily precede x in C (and x itself). This motivates the following definition.

▶ **Definition 2.3** (history). *Let E be an event structure, let $C \in \mathit{Conf}(\mathsf{E})$ and let $x \in C$. The history of x in C is defined as the set $C[x] = \{y \in C \mid y \leq_C x\}$ endowed with the restriction of $\leq_C$ to $C[x]$, i.e., $\leq_{C[x]} = \leq_C \cap (C[x] \times C[x])$. The set of histories in E is $\mathit{Hist}(\mathsf{E}) = \{C[x] \mid C \in \mathit{Conf}(\mathsf{E}) \ \wedge \ x \in C\}$. The set of histories of a specific event $x \in \mathsf{E}$ will be denoted by $\mathit{Hist}(x)$.*

We mentioned that various generalisations of PESs in the literature can be naturally viewed as subclasses of poset event structures. Verifying that the corresponding families of configurations satisfy the properties of Definition 2.2 is easy. We briefly discuss prime and flow event structures (more details are in [6], where also other models are discussed).

Prime event structures. Prime event structures [24] capture the dependencies between events in terms of causality and conflict.

▶ **Definition 2.4** (prime event structure). *A* prime event structure *(PES, for short) is a tuple $\mathsf{P} = \langle E, \leq, \#, \lambda \rangle$, where E is a set of events, $\leq$ and $\#$ are binary relations on E called* causality *and* conflict*, respectively, and $\lambda : E \to \Lambda$ is a labelling function, such that*

- *$\leq$ is a partial order and $\lfloor x \rfloor = \{y \in E \mid y \leq x\}$ is finite for all $x \in E$;*
- *$\#$ is irreflexive, symmetric and hereditary with respect to causality, i.e., for all $x, y, z \in E$, if $x \# y$ and $y \leq z$ then $x \# z$.*

Configurations are sets of events without conflicts and closed with respect to causality. For later use, we also introduce a notation for the absence of conflicts, referred to as consistency.

▶ **Definition 2.5** (consistency, configuration). *Given a PES $\mathsf{P} = \langle E, \leq, \#, \lambda \rangle$, say that $x, y \in E$ are* consistent*, written $x \frown y$, when $\neg(x \# y)$. A subset $X \subseteq E$ is called* consistent*, written $\frown X$, when its elements are pairwise consistent. A* configuration *of P is a finite set of events $C \subseteq E$ such that (i) $\frown C$ and (ii) for all $x \in C$, $\lfloor x \rfloor \subseteq C$.*

Some examples of PESs can be found in Fig. 2. Causality is represented as a solid arrow, while conflict is represented as a dotted line. For instance, in P_0, event a_1 is a cause of b_1 and it is in conflict both with a_2 and b_3. Only direct causalities and non-inherited conflicts are represented. For instance, in P_0, the conflicts $a_1 \# b_2$, $a_2 \# b_1$ and $b_1 \# b_2$ are not represented since they are inherited. The labelling is implicitly represented by naming the events by their label, possibly with some index. For instance, a_1 and a_2 are events labelled by a.

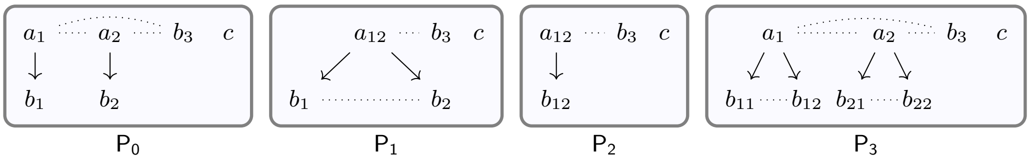


Figure 2 Some prime event structures.

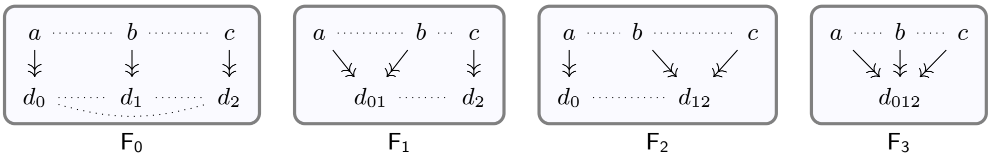


Figure 3 Some flow structures.

Clearly PESs can be seen as poset event structures. Given a PES $\mathsf{P} = \langle E, \leq, \#, \lambda\rangle$ and its set of configurations $\mathit{Conf}(\mathsf{P})$, the local order of a configuration $C \in \mathit{Conf}(\mathsf{P})$ is $\leq_C = \leq \cap (C \times C)$, i.e., the restriction of the causality relation to C. The extension order turns out to be simply subset inclusion. In fact, given $C_1 \subseteq C_2$, if $x_1 \in C_1$ and $x_2 \in C_2$, with $x_2 \leq_{C_2} x_1$, then necessarily $x_2 \in C_1$ since configurations are causally closed. Therefore, recalling that $\leq_{C_1} = \leq \cap (C_1 \times C_1)$ and $\leq_{C_2} = \leq \cap (C_2 \times C_2)$, we immediately conclude that $\leq_{C_2} \cap (C_2 \times C_1) = \leq \cap (C_2 \times C_1) = \leq \cap (C_1 \times C_1) = \leq_{C_1}$, as desired. As an example, the PES P_2 of Fig. 2, viewed as a poset event structure, can be found in Fig. 4.

Flow event structures. Flow event structures [9, 8] extend PESs with the possibility of modelling in a direct way multiple disjunctive and mutually exclusive causes for an event.

▶ **Definition 2.6** (flow event structure). *A* flow event structure *(FES) is a tuple $\langle E, \prec, \#, \lambda\rangle$, where E is a set of events, $\prec \subseteq E \times E$ is an irreflexive relation called the* flow relation, *$\# \subseteq E \times E$ is the* symmetric conflict *relation, and $\lambda : E \to \Lambda$ is a labelling function.*

Causality is replaced by an irreflexive (in general non transitive) flow relation $\prec$, intuitively representing immediate causal dependency. Moreover, conflict is no longer hereditary.

An event can have causes which are in conflict and these have a disjunctive interpretation, i.e., the event will be enabled by a maximal conflict-free subset of its causes.

▶ **Definition 2.7** (FES configuration). *Given a FES $\mathsf{F} = \langle E, \prec, \#, \lambda\rangle$, a configuration is a finite set of events $C \subseteq E$ such that (i) $\prec$ is acyclic on C, (ii) $\neg(x\#x')$ for all $x, x' \in C$ and (iii) for all $x \in C$ and $y \notin C$ with $y \prec x$, there exists $z \in C$ such that $y\#z$ and $z \prec x$.*

Some examples of FESs can be found in Fig. 3. Relation $\prec$ is represented by a double headed solid arrow. For instance, consider the FES F_1. The set $C = \{a, d_{01}\}$ is a configuration. We have $b \prec d_{01}$ and $b \notin C$, but this is fine since there is $a \in C$ such that $a\#b$ and $a \prec d_{01}$.

Under mild assumptions that exclude the presence of non-executable events (a condition referred to as fullness in [8]), FESs can be seen as poset event structures, by endowing each configuration C with a local order arising as the reflexive and transitive closure of the restriction of the flow relation to C, i.e., $\leq_C = (\prec \cap (C \times C))^*$.

2.2 Hereditary History Preserving Bisimilarity

In order to define hereditary history preserving bisimilarity, it is convenient to have an explicit representation of the transitions between configurations.

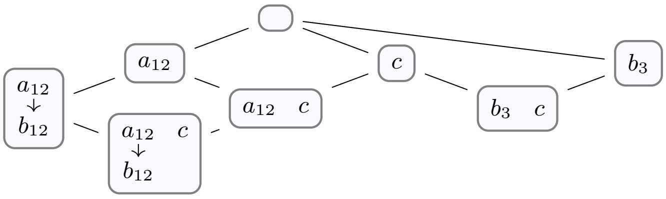


Figure 4 The configurations $Conf(\mathsf{P}_2)$ of the PES P_2 in Fig. 2 viewed as poset event structures.

▶ **Definition 2.8** (transition system). *Let* E *be an event structure. If* $C, C' \in Conf(\mathsf{E})$ *with* $C \sqsubseteq C'$ *we write* $C \xrightarrow{X} C'$ *where* $X = C' \setminus C$.

When X is a singleton, i.e., $X = \{x\}$, we will often write $C \xrightarrow{x} C'$ instead of $C \xrightarrow{\{x\}} C'$.

As it happens in the interleaving approach, a bisimulation between two event structures requires any event of an event structure to be simulated by an event of the other, with the same label. Additionally, the two events must have the same "causal history".

▶ **Definition 2.9** (hereditary history preserving bisimilarity). *Let* E, E' *be event structures. A* hereditary history preserving (hhp-)bisimulation *is a set* R *of triples* (C, f, C'), *where* $C \in Conf(\mathsf{E})$, $C' \in Conf(\mathsf{E}')$ *and* $f : C \to C'$ *is an isomorphism of configurations, such that* $(\emptyset, \emptyset, \emptyset) \in R$ *and for all* $(C_1, f, C_1') \in R$

1. *for all* $C_1 \xrightarrow{x} C_2$ *there exists some* $C_1' \xrightarrow{x'} C_2'$ *such that* $(C_2, f[x \mapsto x'], C_2') \in R$;
2. *for all* $C_1' \xrightarrow{x'} C_2'$ *there exists some* $C_1 \xrightarrow{x} C_2$ *such that* $(C_2, f[x \mapsto x'], C_2') \in R$;
3. *if* $C_2 \in Conf(\mathsf{E})$ *with* $C_2 \sqsubseteq C_1$ *then* $(C_2, f_{|C_2}, f(C_2)) \in R$ *(downward closure).*

Observe that, in the definition above, an event must be simulated by an event with the same label. In fact, in the triple $(C \cup \{x\}, f[x \mapsto x'], C' \cup \{x'\}) \in R$, the second component $f[x \mapsto x']$ must be an isomorphism of configurations, i.e., of labelled posets, and thus it preserves labels. Hhp-bisimilarity has been shown to arise as a canonical behavioural equivalence on prime event structures, as an instance of a general notion defined in terms of the concept of open map, when considering partially ordered computations as observations [18].

3 Foldings of Event Structures

In this section, we study a notion of folding, which is intended to formalise the intuition of a behaviour-preserving quotient for an event structure. We prove that there always exists a minimal quotient and we show that foldings between general poset event structures always arise, in a suitable formal sense, from foldings over prime event structures.

3.1 Morphisms and Foldings

We first endow event structures with a notion of morphism. In the sequel, given two event structures E, E', a function $f : E \to E'$ and a configuration $C \in Conf(\mathsf{E})$, we write $f(C)$ to refer to the poset whose underlying set is $\{f(x) \mid x \in C\}$, endowed with the order $f(x) \leq_{f(C)} f(y)$ iff $x \leq y$.

▶ **Definition 3.1** (morphism). *Let* E, E' *be event structures. A* (strong) morphism $f : \mathsf{E} \to \mathsf{E}'$ *is a function* $f : E \to E'$ *between the underlying sets of events such that* $\lambda = \lambda' \circ f$ *and for all configurations* $C \in Conf(\mathsf{E})$, *the function* f *is injective on* C *and* $f(C) \in Conf(\mathsf{E}')$.

Hereafter, the qualification "strong" will be omitted since this is the only kind of morphisms we deal with. It is motivated by the fact that normally morphisms on event structures are designed to represent simulations. If this were the purpose, then the requirement on preservation of configurations could have been weaker, i.e., we could have asked the order in the target configuration to be included in (not identical to) the image of the order of the source configuration and morphisms could have been partial. However, in our setting, for the objective of defining history-preserving quotients, the stronger notion works fine and simplifies the presentation.

▶ Remark 3.2. *The composition of morphisms is a morphism and the identity is a morphism. Hence the class of event structures and event structure morphisms form a category* $\mathbf{ES}$*.*

▶ **Definition 3.3** (folding)**.** *Let* E *and* E' *be event structures. A* folding *is a morphism* $f : \mathsf{E} \to \mathsf{E}'$ *such that the relation* $R_f = \{(C, f_{|C}, f(C)) \mid C \in \mathit{Conf}(\mathsf{E})\}$ *is a hhp-bisimulation.*

In words, a folding is a function that "merges" some sets of events of an event structure into single events without altering the behaviour modulo hhp-bisimilarity. In [2] the notion of folding is given by requiring the preservation of hp-bisimilarity, a weaker behavioural equivalence defined as hhp-bisimilarity but omitting the requirement of downward-closure (condition 3 in Definition 2.9). Note that, as far as the notion of folding is concerned, this makes no difference: R_f is downward-closed by definition, hence it is a hhp-bisimulation whenever it is a hp-bisimulation. Instead, taking hhp-bisimilarity as the reference equivalence appears to be the right choice for the development of the theory. For instance, it allows one to prove Lemma 3.12 that plays an important role for arguing about the adequateness of the notion of folding (e.g., it is essential for Proposition 3.13). Interestingly, foldings can be characterised as open maps in the sense of [18], by taking conflict free prime event structures as subcategory of observations. This is explicitly worked out in [6].

As an example, consider the PESs in Fig. 2 and the function $f_{02} : \mathsf{P}_0 \to \mathsf{P}_2$ that maps events as suggested by the indices, i.e., $f_{02}(a_1) = f_{02}(a_2) = a_{12}$, $f_{02}(b_1) = f_{02}(b_2) = b_{12}$, $f_{02}(b_3) = b_3$ and $f_{02}(c) = c$. It is easy to see that f_{02} is a folding. Note that, instead, $f_{01} : \mathsf{P}_0 \to \mathsf{P}_1$, again mapping events according to their indices, is not a folding. In fact, $f_{01}(\{a_1\}) = \{a_{12}\} \xrightarrow{b_2} \{a_{12}, b_2\}$, but clearly there is no transition $\{a_1\} \xrightarrow{x}$ with $f_{01}(x) = b_2$, since the only preimage of b_2 in P_0 is b_2.

Observe that the greater expressiveness of FESs allows one to obtain smaller quotients. For instance, consider Fig. 3. The FES F_0 seen as a PES would be minimal. Instead, in the class of FESs it is not: the obvious functions from F_0 to F_1 and F_2 are foldings.

▶ Remark 3.4. *The composition of foldings is a folding and the identity is a folding. We can consider a subcategory* $\mathbf{ES_f}$ *of* $\mathbf{ES}$ *with the same objects and foldings as morphisms.*

Consider again the PESs in Fig. 2 and the morphisms $f_{30} : \mathsf{P}_3 \to \mathsf{P}_0$ and $f_{02} : \mathsf{P}_0 \to \mathsf{P}_2$. These are induced by the labelling, apart for the b_{ij} for which we let $f_{30}(b_{ij}) = b_i$. Both are foldings: the first merges b_{11} with b_{12} and b_{21} with b_{22}, while the second merges a_1 with a_2 and b_1 with b_2. Their composition $f_{32} = f_{30} \circ f_{02} : \mathsf{P}_3 \to \mathsf{P}_2$ is again a folding.

A simple but crucial result shows that the target event structure for a folding is completely determined by the mapping on events. We first define the quotient induced by a morphism.

▶ **Definition 3.5** (quotients from morphisms)**.** *Let* E*,* E' *be event structures and let* $f : \mathsf{E} \to \mathsf{E}'$ *be a morphism. Let* $\equiv_f$ *be the equivalence relation on* E *defined by* $x \equiv_f y$ *if* $f(x) = f(y)$*. We denote by* $\mathsf{E}_{/\equiv_f}$ *the event structure with configurations* $\mathit{Conf}(\mathsf{E}_{/\equiv_f}) = \{[C]_{\equiv_f} \mid C \in \mathit{Conf}(\mathsf{E})\}$ *where* $[C]_{\equiv_f} = \{[x]_{\equiv_f} \mid x \in C\}$ *is ordered by* $[x]_{\equiv_f} \leq_{[C]_{\equiv_f}} [y]_{\equiv_f}$ *iff* $x \leq_C y$*.*

It is immediate to see that $\mathsf{E}_{/\equiv_f}$ is a well-defined event structure.

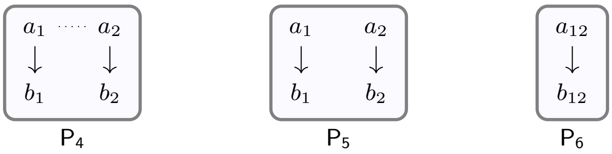


Figure 5 Non existence of pushout of general morphisms.

▶ **Lemma 3.6** (folding as equivalences). *Let* E, E' *be event structures and let* $f : \mathsf{E} \to \mathsf{E}'$ *be a morphism. If* f *is a folding then* $\mathsf{E}_{/\equiv_f}$ *is isomorphic to* E'*.*

The previous result allows us to identify foldings with the corresponding equivalences on the source event structures and motivates the following definition.

▶ **Definition 3.7** (folding equivalences). *Let* E *be an event structure. The set of folding equivalences over* E *is* $\mathit{FEq}(\mathsf{E}) = \{\equiv_f | \ f : \mathsf{E} \to \mathsf{E}' \text{ folding for some } \mathsf{E}'\}$*.*

Hereafter, we will freely switch between the two views of foldings as morphisms or as equivalences, since each will be convenient for some purposes.

We next observe that given two foldings we can always take their "join", providing a new folding that, roughly speaking, produces a quotient smaller than both the original ones.

▶ **Proposition 3.8** (joining foldings). *Let* $\mathsf{E}, \mathsf{E}', \mathsf{E}''$ *be event structures and let* $f' : \mathsf{E} \to \mathsf{E}'$, $f'' : \mathsf{E} \to \mathsf{E}''$ *be foldings. Define* E''' *as the quotient* $\mathsf{E}_{/\equiv}$ *where* $\equiv$ *is the transitive closure of* $\equiv_{f'} \cup \equiv_{f''}$*. Then* $g' : \mathsf{E}' \to \mathsf{E}'''$ *defined by* $g'(x') = [x]_\equiv$ *if* $f'(x) = x'$ *and* $g'' : \mathsf{E}'' \to \mathsf{E}'''$ *defined by* $g''(x'') = [x]_\equiv$ *if* $f''(x) = x''$ *are foldings.*

As an example, consider the PES in Fig. 2 and two morphisms $f_{30} : \mathsf{P}_3 \to \mathsf{P}_0$ and $f_{31} : \mathsf{P}_3 \to \mathsf{P}_1$. The way all events are mapped by f_{30} and f_{31} is naturally suggested by their labelling, apart for the b_{ij} for which we let $f_{30}(b_{ij}) = b_i$ while $f_{31}(b_{ij}) = b_j$. It can be seen that both are foldings. Their join, constructed as in Proposition 3.8, is P_2 with the folding morphisms $f_{02} : \mathsf{P}_0 \to \mathsf{P}_2$ and $f_{12} : \mathsf{P}_1 \to \mathsf{P}_2$.

▶ Remark 3.9. *Proposition 3.8 is a consequence of the fact that the category* **ES** *has pushouts of foldings. Indeed,* E''' *as defined above is the pushout of* f' *and* f'' *(in* **ES** *and also in* $\mathbf{ES_f}$*). It can be seen that, in general,* **ES** *does not have all pushouts.*

As a counterexample to the existence of pushouts in **ES** for general morphisms, consider the obvious mappings $f_{45} : \mathsf{P}_4 \to \mathsf{P}_5$ and $f_{46} : \mathsf{P}_4 \to \mathsf{P}_6$ in Fig. 5. It is easy to realise that, if a pushout existed, the mapping from P_5 into the pushout object should identify the concurrent events a_1 and a_2, failing to be an event structure morphism.

When interpreted in the setting of folding equivalences of an event structure, Proposition 3.8 has a clear meaning. Recall that the equivalences over some fixed set X, ordered by inclusion, form a complete lattice, where the top element is the universal equivalence $X \times X$ and the bottom is the identity on X. Then Proposition 3.8 implies that $\mathit{FEq}(\mathsf{E})$ is a sublattice of the lattice of equivalences. Actually, it can be shown that $\mathit{FEq}(\mathsf{E})$ is itself a complete lattice. Therefore each event structure E admits a maximally folded version.

▶ **Theorem 3.10** (lattice of foldings). *Let* E *be an event structure. Then* $\mathit{FEq}(\mathsf{E})$ *is a sublattice of the complete lattice of equivalence relations over* E*.*

▶ Remark 3.11. *The above result arises from a generalisation of Proposition 3.8 showing that, for any event structure* E*, each collection of foldings* $f_i : \mathsf{E} \to \mathsf{E}_i$*, with* $i \in I$*, admits a colimit in* **ES**. *Thus the coslice category* $(\mathsf{E} \downarrow \mathbf{ES_f})$ *has a terminal object, which is the maximally folded event structure.*

It is natural to ask whether behaviour-preserving quotients correspond to foldings. Strictly speaking, the answer is negative. More precisely, there can be morphisms $f : \mathsf{E} \to \mathsf{E}'$ such that $\mathsf{E}_{/\equiv_f}$ is hhp-bisimilar to E, but f is not a folding. For an example, consider the PESs P_0 and P_1 in Fig. 2 and the morphism $f_{01} : \mathsf{P}_0 \to \mathsf{P}_1$ suggested by the indexing. We already observed this is not a folding, but $\mathsf{P}_{0/\equiv_{f_{01}}}$, which is isomorphic to P_1, is hhp-bisimilar to P_0. However, we can show that for any behaviour-preserving quotient, there is a folding that produces a coarser equivalence, and thus a smaller quotient. For instance, in the example discussed above, there is the folding $f_{02} : \mathsf{P}_0 \to \mathsf{P}_2$, that "produces" a smaller quotient.

This follows from the possibility of joining foldings (Proposition 3.8) and the fact that a hhp-bisimulation can be always seen as an event structure, a result that generalises to our setting a property proved for PESs in [7].

▶ **Lemma 3.12** (hhp-bisimulation as an event structure). *Let E', E'' be event structures and let R be a hhp-bisimulation between them. Then there exists a (prime) event structure E_R and two foldings $\pi' : \mathsf{E}_R \to \mathsf{E}'$ and $\pi'' : \mathsf{E}_R \to \mathsf{E}''$.*

▶ **Proposition 3.13** (foldings subsume behavioural quotients). *Let E be an event structure and let $f : \mathsf{E} \to \mathsf{E}'$ be a morphism such that $\mathsf{E}_{/\equiv_f}$ is hhp-bisimilar to E. Then there exists a folding $g : \mathsf{E} \to \mathsf{E}''$ such that $\equiv_g$ is coarser than $\equiv_f$.*

The proof relies on the possibility of joining foldings (Proposition 3.8) and the fact that a hhp-bisimulation can be always seen as an event structure, a result that generalises to our setting a property proved for PESs in [7].

3.2 Folding through Prime Event Structures

We observe that each event structure is the folding of a corresponding canonical PES. We then prove that, interestingly enough, all foldings between event structures arise from foldings of the corresponding canonical PESs.

We start with the definition of the canonical PES associated with an event structure.

▶ **Definition 3.14** (PES for an event structure). *Let E be an event structure. Its canonical PES is $\mathbb{P}(\mathsf{E}) = \langle \mathit{Hist}(\mathsf{E}), \sqsubseteq, \#, \lambda' \rangle$ where $\sqsubseteq$ is prefix, $\#$ is incompatibility, i.e., for $H_1, H_2 \in \mathit{Hist}(\mathsf{E})$ we let $H_1 \# H_2$ if $\neg(H_1 \frown H_2)$, and $\lambda'(H) = \lambda(x)$ when $H \in \mathit{Hist}(x)$. Given a morphism $f : \mathsf{E} \to \mathsf{E}'$ we write $\mathbb{P}(f) : \mathbb{P}(\mathsf{E}) \to \mathbb{P}(\mathsf{E}')$ for the morphism defined by $\mathbb{P}(f)(H) = f(H)$.*

It can be easily seen that the definition above is well-given. In particular, $\mathbb{P}(\mathsf{E})$ is a well-defined PES because, as proved in [26], a family of posets ordered by prefix is finitary coherent prime algebraic domain. Then the tight relation between this class of domains and PES highlighted in [33] allows one to conclude the proof. For instance, in Fig. 1(right) one can find the canonical PES for the event structure on the left.

The canonical PES associated with an event structure can always be folded to the original event structure.

▶ **Lemma 3.15** (unfolding event structures to PES's). *Let E be an event structure. Define a function $\phi_{\mathsf{E}} : \mathbb{P}(\mathsf{E}) \to \mathsf{E}$, for all $H \in \mathit{Hist}(\mathsf{E})$ by $\phi_{\mathsf{E}}(H) = x$ if $H \in \mathit{Hist}(x)$ for $x \in \mathsf{E}$. Then ϕ_{E} is a folding.*

We next show that any morphism from a PES to an event structure E factorises uniquely through the PES $\mathbb{P}(\mathsf{E})$ associated with E (categorically, ϕ_{E} is cofree over E). The same applies to foldings and it will be useful to relate foldings in E with foldings in $\mathbb{P}(\mathsf{E})$.

▶ **Lemma 3.16** (cofreeness of ϕ_{E}). *Let E be an event structure, let P' be a* PES *and let $f : \mathsf{P}' \to \mathsf{E}$ be an event structure morphism. Then there exists a unique morphism $g : \mathsf{P}' \to \mathbb{P}(\mathsf{E})$ such that $f = \phi_{\mathsf{E}} \circ g$.*

$$\begin{tikzcd}
\mathbb{P}(\mathsf{E}) \arrow[r, "\phi_{\mathsf{E}}"] & \mathsf{E} \\
\mathsf{P}' \arrow[u, dotted, "g"] \arrow[ur, "f"'] &
\end{tikzcd}$$

Moreover, when f is a folding then so is g.

▶ Remark 3.17. *Lemma 3.16 means that the category $\mathbf{PES}$ of prime event structures is a coreflective subcategory of $\mathbf{ES}$, i.e., $\mathbb{P} : \mathbf{ES} \to \mathbf{PES}$ can be seen as a functor, right adjoint to the inclusion $\mathbb{I} : \mathbf{PES} \to \mathbf{ES}$. Moreover, $\mathbb{P}$ restricts to a functor on the subcategory of foldings, $\mathbb{P} : \mathbf{ES_f} \to \mathbf{PES_f}$, where an analogous result holds.*

We conclude that all foldings between event structures arise from foldings of the associated PESs. Given that $\mathbf{PES}$ is a coreflective subcategory of $\mathbf{ES}$ and foldings can be seen as open maps, this result (and also the fact that morphisms ϕ_{E} are foldings) can be derived from [18, Lemma 6]. More details on this can be found in the extended version [6].

▶ **Proposition 3.18** (folding through PES's). *Let E, E' be event structures. For all morphisms $f : \mathsf{E} \to \mathsf{E}'$ consider $\mathbb{P}(f) : \mathbb{P}(\mathsf{E}) \to \mathbb{P}(\mathsf{E}')$ defined by $\mathbb{P}(f)(H) = f(H)$. Then f is a folding iff $\mathbb{P}(f)$ is a folding.*

4 Foldings for Prime Event Structures

Motivated by the fact that foldings on general poset event structures always arise from foldings of the corresponding canonical PESs, in this section we study foldings in the class of PESs. We provide a characterisation and show that also PESs always admit a least quotient.

Since foldings are special morphisms, we first provide a characterisation of PES morphisms.

▶ **Lemma 4.1** (PES morphisms). *Let P and P' be PESs and let $f : P \to P'$ be a function on the underlying sets of events. Then f is a morphism iff for all $x, y \in \mathsf{P}$*

1. *$\lambda'(f(x)) = \lambda(x)$;*
2. *$f(\lfloor x \rfloor) = \lfloor f(x) \rfloor$; namely (a) for all $z' \in \mathsf{P}'$, if $z' \leq f(x)$ there exists $z \in \mathsf{P}$ such that $z \leq x$ and $f(z) = z'$ (b) if $z \leq x$ then $f(z) \leq f(x)$;*
3. *(a) if $f(x) = f(y)$ and $x \neq y$ then $x \# y$ and (b) if $f(x) \# f(y)$ then $x \# y$.*

These are the standard conditions characterising (total) PES morphisms (see, e.g., [33]), with the addition of condition (2b) that is imposed to ensure that configurations are mapped to isomorphic configurations, as required by the notion of (strong) morphism (Definition 3.1).

We know that not all PES morphisms are foldings. We next identify some additional conditions characterising those morphisms which are foldings. Given a relation $r \subseteq X \times X$ and $Y, Z \subseteq X$ we write $Y \ r^{\forall} \ Z$ if for all $y \in Y$ and $z \in Z$ it holds $y \ r \ z$. Singletons are replaced by their only element, writing, e.g., $y \ r^{\forall} \ Z$ for $\{y\} \ r^{\forall} \ Z$.

▶ **Theorem 4.2** (PES foldings). *Let P and P' be PESs and let $f : \mathsf{P} \to \mathsf{P}'$ be a morphism. Then f is a folding if and only if it is surjective and for all $X, Y \subseteq \mathsf{P}$, $x, y \in \mathsf{P}$, $y' \in \mathsf{P}'$*

1. *if $x \#^{\forall} f^{-1}(y')$ then $f(x) \# y'$;*
2. *if $\frown(X \cup \{x\})$, $\frown(Y \cup \{y\})$, $\frown(X \cup Y)$ and $f(x) = f(y)$ then there exists $z \in \mathsf{P}$ such that $f(z) = f(x)$ and $\frown(X \cup Y \cup \{z\})$.*

The notion of folding on PESs turns out to be closely related to that of abstraction homomorphism for PESs introduced in [12] for similar purposes. More precisely, abstraction homomorphisms can be characterised as those PES morphisms additionally satisfying condition (1) of Theorem 4.2, while they do not necessarily satisfy condition (2). Their more liberal definition with respect to foldings can be explained by the fact that they are designed to work on a subclass of structured PESs (a formal comparison is in the extended version [6]).

The next result "transfers" the conditions characterising foldings to folding equivalences.

▶ **Corollary 4.3** (folding equivalences for PES's). *Let* P *be a* PES *and let* $\equiv$ *be an equivalence on* P*. Then* $\equiv$ *is a folding equivalence in* $FEq(\mathsf{P})$ *iff for all* $x, y \in \mathsf{P}$*,* $x \neq y$*, if* $x \equiv y$ *then*

1. $\lambda(x) = \lambda(y)$*;*
2. $[\lfloor x \rfloor]_{\equiv} = [\lfloor y \rfloor]_{\equiv}$*;*
3. $x \# y$*.*

Moreover, for all $x, y \in \mathsf{P}$*,* $X, Y \subseteq \mathsf{P}$

4. *if* $x \#^{\forall} [y]_{\equiv}$ *then* $[x]_{\equiv} \#^{\forall} [y]_{\equiv}$*;*
5. *if* $\frown(X \cup \{x\})$*,* $\frown(Y \cup \{y\})$*,* $\frown(X \cup Y)$ *there exists* $z \in [x]_{\equiv}$ *such that* $\frown(X \cup Y \cup \{z\})$*.*

For instance, in Fig. 2, consider the equivalence $\equiv_{01}$ over P_0 such that $a_1 \equiv_{01} a_2$. This produces P_1 as quotient. This is not a folding equivalence since condition (4) fails: $a_1 \#^{\forall} [b_2]_{\equiv_{01}}$, but $\neg(a_2 \# b_2)$ and thus $\neg([a_1]_{\equiv_{01}} \#^{\forall} [b_2]_{\equiv_{01}})$. Instead, the equivalence $\equiv_{02}$ over P_0 such that $a_1 \equiv_{02} a_2$ and $b_1 \equiv_{02} b_2$, producing P_2 as quotient, satisfies all five conditions.

When PESs are finite, the result above suggests a possible way of identifying foldings: one can pair candidate events to be folded on the basis of conditions (1)-(3) and then try to extend the sets with condition (4)-(5) when possible. The procedure can be inefficient due to the global flavor of the conditions. This will be further discussed in the conclusions.

We know, from Proposition 3.8, that all event structures admit a "maximally folded" version. We next observe that the same result continues to hold in the class of PESs.

▶ **Theorem 4.4** (joining foldings on PES's). *Let* $\mathsf{P}, \mathsf{P}', \mathsf{P}''$ *be* PES*s and let* $f' : \mathsf{P} \to \mathsf{P}'$*,* $f'' : \mathsf{P} \to \mathsf{P}''$ *be foldings. Define* E''' *along with* $g' : \mathsf{P}' \to \mathsf{E}'''$ *and* $g'' : \mathsf{P}'' \to \mathsf{E}'''$ *as in Proposition 3.8. Then* E''' *is a* PES*. As a consequence, each* PES *admits a uniquely determined minimal quotient in the class of* PES*s.*

▶ Remark 4.5. *Theorem 4.4 is a consequence of the fact that the subcategory* $\mathbf{PES_f}$ *is a coreflective subcategory of* $\mathbf{ES_f}$ *and thus it is closed under colimits.*

In passing, we note that in the class of FESs the existence of a unique minimal folding is lost. In fact, consider Fig. 3. It can be easily seen that F_1 and F_2 are different minimal foldings of F_0. In particular, merging the three d-labelled events as done in F_3 modifies the behaviour. In fact, in F_3, the event d_{012} is not enabled in $C = \{a\}$ since $c \prec d_{012}$ and no event in C is in conflict with c. Instead, in F_0, the event d_0 is clearly enabled from $\{a\}$. Existence of a unique minimal folding could be possibly recovered by strengthening the notion of folding. Note, however, that this would be against the spirit of our work where the notion of folding is not a choice. Rather, after having assumed hhp-bisimilarity as the reference behavioural equivalence, the notion of folding is essentially "determined" as a quotient (surjective function) that preserves the behaviour up to hhp-bisimilarity.

5 Conclusions

We studied the problem of minimisation for poset event structures, a class that encompasses many stable event structure models in the literature, taking hereditary history preserving bisimilarity as reference behavioural equivalence. We showed that a uniquely determined minimal quotient always exists for poset event structures and also in the subclass of PESs. We showed that foldings between general poset event structures arise from foldings of corresponding canonical PESs. Finally, we provided a characterisation of foldings of PESs.

We believe that, besides its original motivations from the setting of business process models and its foundational interest, this work can be of help in the study of minimisation, under a true concurrent equivalence, of operational models which can be mapped to event structures, like transition systems with independence or Petri nets.

As underlined throughout the paper, our theory of folding has many connections with the literature on event structures. The idea of "unfolding" more expressive models to prime algebraic domains and PESs has been studied by many authors (e.g., in [26, 24, 28, 30, 9]). The same can be said for the idea of refining a single action into a complex computation (see, e.g., [29] and references therein). Instead, the problem of characterising behaviour-preserving quotients of event structures has received less attention. We already commented on the relation with abstraction homomorphisms for PESs [12], which capture the idea of behaviour-preserving abstraction but only in a subclass of structured PESs. In some cases, given a Petri net or an event structure a special transition system can be extracted, on which minimisation is performed. In particular, the paper [23] proposes an encoding of safe Petri nets into causal automata, preserving hp-bisimilarity. Such automata can be transformed into a standard labelled transition systems, which in turn can be minimised. However, in this way, the correspondence with the original events is lost.

The notion of behaviour-preserving function has been given an elegant abstract characterisation in terms of open maps [18]. We mentioned the possibility of viewing our foldings as open maps and we observed that various results admit a categorical interpretation (see also [6]). This gives clear indications of the possibility of providing a general abstract view of the results in this paper, something which represents an interesting topic of future research.

The characterisation of foldings on PESs can be used as a basis to develop, at least in the case of finite structures, algorithms for the construction of behaviour preserving quotients. The fact that conditions for folding refer to sets of events might make the minimisation procedure very inefficient. Identifying suitable heuristics and investigating the possibility of having more "local" folding conditions are interesting directions of future work.

Although not explicitly discussed in the paper, by considering elementary foldings, i.e., foldings that just merge a single set of events, one can indeed determine some more efficient folding rules. This is essentially what is done for AESs and FESs in [2]. However, restricting to elementary foldings is limiting, since it can be seen that general foldings cannot be always decomposed in terms of elementary ones (e.g., it can be seen that in Fig. 2, the folding $f_{02} : \mathsf{P}_0 \to \mathsf{P}_2$ cannot be obtained as the composition of elementary foldings).

When dealing with possibly infinite event structures one could try to devise reduction rules acting on some finitary representation and inducing foldings on the corresponding event structure. Note, however, that working, e.g., on finite safe Petri nets, the minimisation procedure would be necessarily incomplete, given that hhp-bisimilarity is undecidable [19].

References

1 A. Armas-Cervantes, P. Baldan, M. Dumas, and L. García-Bañuelos. Diagnosing behavioral differences between business process models: An approach based on event structures. *Information Systems*, 56:304–325, 2016.

2 A. Armas-Cervantes, P. Baldan, and L. García-Bañuelos. Reduction of event structures under history preserving bisimulation. *Journal of Logical and Algebraic Methods in Programming*, 85(6):1110–1130, 2016.

3 P. Baldan. *Modelling concurrent computations: from contextual Petri nets to graph grammars*. PhD thesis, University of Pisa, 2000.

4 P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent Semantics of Algebraic Graph Transformation Systems. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume III: Concurrency, pages 107–187. World Scientific, 1999.

5 P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures and processes. *Information and Computation*, 171(1):1–49, 2001.

6 P. Baldan and A. Raffaetà. Minimising Event Structures, 2019. arXiv:1907.07042.

7 M.A. Bednarczyk. Hereditary History Preserving Bisimulations or What is the Power of the Future Perfect in Program Logics. Technical report, Polish Academy of Sciences, 1991.

8 G. Boudol. Flow Event Structures and Flow Nets. In *Semantics of System of Concurrent Processes*, volume 469 of *LNCS*, pages 62–95. Springer Verlag, 1990.

9 G. Boudol and I. Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In *Linear Time, Branching Time and Partial Order Semantics in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 411–427. Springer Verlag, 1988.

10 R. Bruni, H.C. Melgratti, and U. Montanari. Event Structure Semantics for Nominal Calculi. In C. Baier and H. Hermanns, editors, *CONCUR 2006*, volume 4137 of *LNCS*, pages 295–309. Springer, 2006.

11 S. Castellan. Weak memory models using event structures. In *Proceedings of JFLA'16*, 2016.

12 I. Castellani. *Bisimulations for Concurrency*. PhD thesis, University of Edimburgh, 1988.

13 S. Chakraborty and V. Vafeiadis. Grounding thin-air reads with event structures. *PACMPL*, 3(POPL):70:1–70:28, 2019.

14 I. Cristescu, J. Krivine, and D. Varacca. Rigid families for CCS and the pi-calculus. In *Proceedings of ICTAC'15*, volume 9399 of *LNCS*, pages 223–240. Springer, 2015.

15 M. Dumas and L. García-Bañuelos. Process Mining Reloaded: Event Structures as a Unified Representation of Process Models and Event Logs. In R.R. Devillers and A. Valmari, editors, *Petri Nets 2015*, volume 9115 of *LNCS*, pages 33–48. Springer, 2015.

16 J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2006.

17 A. Jeffrey and J. Riely. On Thin Air Reads: Towards an Event Structures Model of Relaxed Memory. In M. Grohe, E. Koskinen, and N. Shankar, editors, *LICS 2016*, pages 759–767. ACM, 2016.

18 A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from Open Maps. *Information and Computation*, 127(2):164–185, 1996.

19 M. Jurdzinski, M. Nielsen, and J. Srba. Undecidability of domino games and hhp-bisimilarity. *Information and Computation*, 184(2):343–368, 2003.

20 R. Langerak. Bundle Event Structures: A Non-Interleaving Semantics for Lotos. In *5th Intl. Conf. on Formal Description Techniques (FORTE'92)*, pages 331–346. North-Holland, 1992.

21 R. Langerak. *Transformation and Semantics for LOTOS*. PhD thesis, Department of Computer Science, University of Twente, 1992.

22 A.R. Meyer and L.J. Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In *SWAT (FOCS)*, pages 125–129. IEEE Computer Society, 1972.

23 U. Montanari and M. Pistore. Minimal Transition Systems for History-Preserving Bisimulation. In *14th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *LNCS*, pages 413–425. Springer Verlag, 1997.

24 M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.

25 J. Pichon-Pharabod and P. Sewell. A concurrency semantics for relaxed atomics that permits optimisation and avoids thin-air executions. In R. Bodík and R. Majumdar, editors, *POPL 2016*, pages 622–633. ACM, 2016.

26 A. Rensink. Posets for Configurations! In W. R. Cleaveland, editor, *Proceedings of CONCUR'92*, volume 630 of *LNCS*, pages 269–285. Springer, 1992.

27 G. Schied. On relating Rewriting Systems and Graph Grammars to Event Structures. In H.-J. Schneider and H. Ehrig, editors, *Dagstuhl Seminar 9301 on Graph Transformations in Computer Science*, volume 776 of *LNCS*, pages 326–340. Springer, 1994.

28 R.J. van Glabbeek. History preserving process graphs. Draft available at `http://theory.stanford.edu/~rvg/abstracts.html#hppg`, 1996.

29 R.J. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4/5):229–327, 2001.

30 R.J. van Glabbeek and G.D. Plotkin. Configuration structures, event structures and Petri nets. *Theoretical Computer Science*, 410(41):4111–4159, 2009.

31 D. Varacca and N. Yoshida. Typed event structures and the linear pi-calculus. *Theoretical Computer Science*, 411(19):1949–1973, 2010.

32 G. Winskel. Event Structure Semantics for CCS and Related Languages. Technical Report DAIMI PB-159, University of Aarhus, 1983.

33 G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 325–392. Springer, 1987.

34 G. Winskel. Events, Causality and Symmetry. *Computer Journal*, 54(1):42–57, 2011.

Concurrent Parameterized Games

Nathalie Bertrand
Univ. Rennes, Inria, CNRS, IRISA, Rennes, France

Patricia Bouyer
LSV, CNRS & ENS Paris-Saclay, Univ. Paris-Saclay, Cachan, France

Anirban Majumdar
Univ. Rennes, Inria, CNRS, IRISA, Rennes, France
LSV, CNRS & ENS Paris-Saclay, Univ. Paris-Saclay, Cachan, France

Abstract

Traditional concurrent games on graphs involve a fixed number of players, who take decisions simultaneously, determining the next state of the game. In this paper, we introduce a parameterized variant of concurrent games on graphs, where the parameter is precisely the number of players. Parameterized concurrent games are described by finite graphs, in which the transitions bear regular languages to describe the possible move combinations that lead from one vertex to another.

We consider the problem of determining whether the first player, say Eve, has a strategy to ensure a reachability objective against any strategy profile of her opponents as a coalition. In particular Eve's strategy should be independent of the number of opponents she actually has. Technically, this paper focuses on an *a priori* simpler setting where the languages labeling transitions only constrain the number of opponents (but not their precise action choices). These constraints are described as semilinear sets, finite unions of intervals, or intervals.

We establish the precise complexities of the parameterized reachability game problem, ranging from PTIME-complete to PSPACE-complete, in a variety of situations depending on the contraints (semilinear predicates, unions of intervals, or intervals) and on the presence or not of non-determinism.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases concurrent games, parameterized verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.31

Funding *Patricia Bouyer*: has been funded by ERC project EQualIS.

Acknowledgements We thank Christoph Haase for insightful discussions on semilinear sets.

1 Introduction

Parameterized verification. The generalisation and everyday usage of, for example, cloud computing and blockchains technology, calls for the verification of algorithms running on distributed systems. Concrete examples are consensus and leader-election algorithms, but also coherence protocols, etc. This explains the recent interest of the model-checking community for the verification of systems composed of an arbitrary number of agents [10, 5].

Verifying algorithms running on distributed systems for all possible number of agents at once calls for symbolic techniques. These are generic, and compare favorably –in terms of complexity– to applying standard verification techniques on a given instance with a fixed large number of agents. Therefore, beyond its original goal of verifying systems independently of the number of agents, parameterized verification can also be more efficient than standard verification for large systems. In the last 15 years, parameterized verification algorithms were successfully applied to various case studies, such as data-consistency for cache coherence protocols in uniform memory access multiprocessors [9], and the core of simple reliable broadcast protocols in asynchronous systems [13].

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 31; pp. 31:1–31:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Multiplayer concurrent games. In parallel, for multi-agents systems, the AI and model-checking communities traditionally use concurrent games on graphs to model the complex interactions between agents [1, 2]. An arena for n players is a directed graph where the transitions are labeled by n-tuples of actions. At each vertex of the graph, the n players select simultaneously and independently an action, and the next vertex is determined by the combined move consisting of all the actions. Most often, one considers infinite duration plays, that is plays generated by iterating this process forever. Concepts studied on multiplayer concurrent games include some borrowed from game theory, such as winning strategies (see e.g. [1]), rationality of players (see e.g. [11]), Nash equilibria (see e.g. [17, 6]).

Concurrent games with a parameterized number of players. The purpose of the current paper is to settle the foundations of concurrent games involving a parameterized number of players, paving the way to the modelling and verification of interactions involving an arbitrary number of agents. We envision that such games may later have applications in a variety of contexts, such as telecommunications and distributed algorithms. The conclusion presents a simple coordination game, and one of our long-term objectives is to solve the distributed synthesis problem of such games.

Generalising concurrent games to a parameterized number of agents can be done by replacing, on edges of the arena, tuples representing the choice of each of the agents by languages of finite yet *a priori* unbounded words. It seems natural to first consider regular languages, represented by regular expressions. For instance the label a^+ represents that all players choose action a, while ab^+ is the situation where the first player chooses a, while all other players play b. Such a parameterized arena can represent infinitely many interaction situations, one for each possible number of agents. In parameterized concurrent games, the agents do not know *a priori* the number of agents participating to the interaction. Each player observes the action it plays and the vertices the play goes through. These pieces of information may refine the knowledge each player has on the number of involved agents.

Figure 1 presents a first example of a parameterized arena. This arena represents a situation where the players need to figure out the parity of their number in order to make a correct decision (action b if there is an even number of players, and c otherwise). Here, players can collectively reach the target vertex v_4: they all play a in the two first steps, and from v_3, if the play went through v_1 (resp. v_2), they all play b (resp. c).

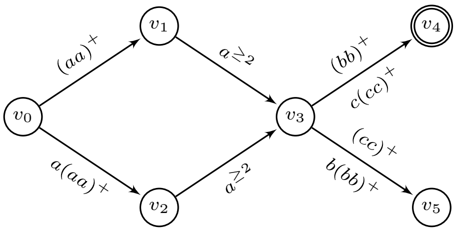


Figure 1 Example of a parameterized arena.

As for traditional concurrent games, one can consider natural questions such as, for instance, the distributed synthesis problem as in the above example, or the existence and computation of Nash equilibria. To start with, we consider a simpler decision problem: the first player, called Eve, is distinguished, and the question is whether she can ensure a reachability objective against the coalition of the other players, not knowing *a priori* the number of her opponents. She therefore must play uniformly, whatever the number of opponents she has. To simplify the exposition, we assume that the languages on transition of the arena are particularly simple: they only constrain the number of opponents Eve has. However, as discussed in Section 4, this simpler setting is not restrictive for the decision problem we consider.

Contributions. After the definition of the parameterized game setting, the main contribution of this paper is the resolution of the so-called parameterized reachability game problem, with tight complexity bounds. We distinguish several cases, depending on whether arenas are deterministic or not, and on whether constraints on the number of opponents are intervals, finite unions of intervals, or semilinear sets.

The existence of a uniformly winning strategy for Eve reduces to the resolution of the *knowledge game*, a two-player reachability turn-based game. The latter is *a priori* exponential in the size of the original arena, since vertices include the knowledge Eve has on the possible number of her opponents, and this exponential blowup is unavoidable. Yet, when constraints are only intervals, the knowledge game is only of polynomial size. In this particular case, we prove the parameterized game problem to be PTIME-complete. For finite unions of intervals, and when the parameterized arena is deterministic, we show that if Eve has a winning strategy, she has one that can be represented by a polynomial size strategy tree. This small model property, together with the encoding of 3SAT allows us to prove the problem to be NP-complete. Finally, for finite unions of intervals and non-deterministic arenas, or for semilinear sets (with no assumption of non-determinism) the parameterized game problem is PSPACE-complete. The lower bound is obtained by a reduction from QBF-SAT, while the upper bound derives from a depth-first search algorithm on an exponential size tree, non-trivially extracted from the knowledge game. All the complexities are summarized in Table 1, on page 6.

Related work. Up to our knowledge, this contribution is the first to introduce and study a model of concurrent games with a parameterized number of players. Our model of parameterized concurrent games mixes interactions and an arbitrary number of agents. As far as we are aware, only a couple of other works in parameterized verification have defined a game semantics, and they all largely differ from the current setting. First, to study broadcast networks of many identical Markov decision processes, broadcast networks of two-player games were introduced [4]. There, the behaviour of each agent is the same and is described by a two-player turn-based game. Second, a control problem for an arbitrary size population of identical agents was studied in [3]. In that work, a controller plays against a parameterized number of agents, similarly to Eve playing against an unknown number of opponents. However, in contrast to our parameterized games, in the population control problem, the semantics is a turn-based game, and, most importantly, the arena is not centralized.

2 Game setting

We first introduce parameterized arenas, which form a simple setting for modelling games with a parameterized number of players. In such arenas, edges are labeled with sets of pairs (a, k) for a an action of Eve, and k a number of opponents. We discuss in Section 4 how a natural extension of concurrent games to a parameterized number of players, with regular languages on edges, reduces to this simpler setting. In the whole paper, we denote by $\mathbb{N}$ the set of natural numbers (including 0) and write $\mathbb{N}_{>0}$ for the set of positive natural numbers.

▶ **Definition 1.** *A* parameterized arena *is a tuple* $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$ *where*

- V *is a finite set of vertices;*
- Σ *is a finite set of actions;*
- $\Delta : V \times \Sigma \times \mathbb{N}_{>0} \to 2^V$ *is the transition function.*

The arena is *deterministic* if for every $v \in V$, and every pair $(a, k) \in \Sigma \times \mathbb{N}_{>0}$, there is at most one vertex $v' \in V$ such that $v' \in \Delta(v, a, k)$. Action $a \in \Sigma$ is *enabled* at vertex v if there exists $k \in \mathbb{N}_{>0}$ such that $\Delta(v, a, k) \neq \emptyset$. The arena is assumed to be *complete for enabled actions*: for every $v \in V$, if a is enabled at v, then for all $k \in \mathbb{N}_{>0}$, $\Delta(v, a, k) \neq \emptyset$. This assumption is natural: Eve does not know how many opponents she has, and the successor vertex must exist whatever that number is. Given a predicate $P \subseteq \mathbb{N}_{>0}$, $\Delta(v, a, P)$ is a shorthand for $\bigcup_{k \in P} \Delta(v, a, k)$.

Further, for any $v, v' \in V$ and $a \in \Sigma$, we introduce the following notation to represent the set of number of opponents that can lead from v to v' under action a of Eve: $\nabla(v, a, v') = \{k \in \mathbb{N}_{>0} \mid v' \in \Delta(v, a, k)\}$. Finally, we write $E = \{(v, a, v') \mid \exists k \in \mathbb{N}_{>0},\ v' \in \Delta(v, a, k)\}$ for the set of *edges* of the arena.

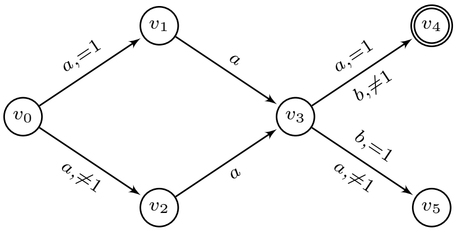


Figure 2 Example of a parameterized reachability game.

▶ **Example 2.** An example of a deterministic parameterized reachability game is presented in Figure 2, with $V = \{v_0, \ldots, v_5\}$, $\Sigma = \{a, b\}$. Here and in other pictures, we use constraints to represent the transition function: for instance, the label '$a, = 1$' on the transition from v_0 to v_1 represents $\Delta(v_0, a, 1) = \{v_1\}$, and the label '$a, \neq 1$' means that for every $k \neq 1$ (that is, $k \geq 2$), $\Delta(v_0, a, k) = \{v_2\}$, or simply $\Delta(v_0, a, \neq 1) = \{v_2\}$. Moreover, we omit the constraint if it is trivial e.g., for every $k \in \mathbb{N}_{>0}$, $\Delta(v_1, a, k) = \{v_3\}$. On that example, action a is the only enabled action at vertices v_0, v_1 and v_2, and both a and b are enabled at v_3. Also (v_0, a, v_1) is an example of edge. Finally, $\nabla(v_3, a, v_4) = \{1\}$ and $\nabla(v_3, b, v_4) = [2, \infty)$.

Let $k \in \mathbb{N}_{>0}$. A *k-history*, for a coalition composed of k opponents of Eve, is a finite sequence $v_0 a_0 \cdots v_i \in (V \cdot \Sigma)^* \cdot V$ such that for every $j < i$, $v_{j+1} \in \Delta(v_j, a_j, k)$ (or equivalently $k \in \bigcap_{j<i} \nabla(v_j, a_j, v_{j+1})$). A *history* in $\mathcal{A}$ is a k-history for some $k \in \mathbb{N}_{>0}$. We note $\mathsf{Hist}(k)$ (resp. Hist) for the set of k-histories (resp. histories) in $\mathcal{G}$. Similar notions of a *k-play* and a *play* are defined for infinite sequences.

▶ **Definition 3.** *A* strategy *for Eve from v in $\mathcal{A}$ is a mapping $\sigma : \mathsf{Hist} \to \Sigma$ that associates to every history $hv' \in \mathsf{Hist}$ an action $\sigma(hv')$ which is enabled at v'. Further, σ is* memoryless *whenever for every $hv', h'v' \in \mathsf{Hist}$, $\sigma(hv') = \sigma(h'v')$.*

A strategy for Eve is applied with no prior information on the number of her opponents. Given a strategy σ, an initial vertex v and $k \in \mathbb{N}_{>0}$ a number of opponents, we define the outcome $\mathsf{Out}(\sigma, v, k)$ as the set of plays that σ induces from v when Eve has exactly k opponents. Formally, $\mathsf{Out}(\sigma, v, k)$ is the set of all k-plays $\rho = v_0 a_0 v_1 a_1 v_2 \cdots$ such that $v = v_0$, and for all $i \geq 0$, $\sigma(v_0 a_0 \cdots v_i) = a_i$ and $v_{i+1} \in \Delta(v_i, a_i, k)$. The completeness assumption ensures that the set $\mathsf{Out}(\sigma, v, k)$ is not empty. Finally, $\mathsf{Out}(\sigma)$ is the set of all possible plays induced by σ from v: $\mathsf{Out}(\sigma, v) = \bigcup_{k \geq 1} \mathsf{Out}(\sigma, v, k)$.

Given an arena $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$, a target vertex $t \in V$ defines a *reachability game* $\mathcal{G} = (\mathcal{A}, t)$ for Eve. A strategy σ for Eve from v in the reachability game $\mathcal{G} = (\mathcal{A}, t)$ is *winning* if all plays in $\mathsf{Out}(\sigma, v)$ eventually reach t. If there exists a winning strategy from v, then we say that v belongs to the *winning region* of Eve.

▶ **Example 4.** Resuming Example 2, one can show that Eve has a winning strategy σ from v_0 to reach the target v_4 defined by $\sigma(v_0) = \sigma(v_0 a v_1) = \sigma(v_0 a v_2) = a$, $\sigma(v_0 a v_1 a v_3) = a$ and $\sigma(v_0 a v_2 a v_3) = b$. Intuitively, the decision at vertex v_3 depends on whether the play went through v_1 –in this case Eve deduces that she has a single opponent– or v_2. Note that no memoryless strategy is winning for Eve: if she always chooses a at v_3, she is losing against more than 1 opponents; and similarly for b. The winning region for Eve is $\{v_0, v_4\}$.

The purpose of this paper is to establish the complexity of the following decision problem:

Parameterized reachability game problem
Input: A parameterized reachability game $\mathcal{G} = (\mathcal{A}, t)$ and an initial vertex v.
Question: Does Eve have a winning strategy from v in $\mathcal{G}$?

For algorithmic reasons, we assume the transition function Δ of $\mathcal{A}$ can be described in a finite way. More precisely, the sets $\nabla(v, a, v')$ for $v, v' \in V$ and $a \in \Sigma$ should be simple enough.

We first consider constraints described by closed intervals (since we deal with sets of natural numbers, it is no restriction to assume intervals to be closed) or finite unions of closed intervals. If $[a, b]$ (resp. $[a, \infty)$) is an interval, then we say a is a left endpoint and b (resp. ∞) is a right endpoint. As a complexity parameter, we use $\#\mathsf{endpoints}_{\mathcal{A}}$, the number of endpoints used in constraints in $\mathcal{A}$. All the complexities will be functions of this parameter, independently of the precise values of the endpoints.

More generally, we also consider semilinear predicates over $\mathbb{N}$. A simple example of a semilinear predicate is the predicate "divisible by p", where $p \in \mathbb{N}_{>0}$. W.l.o.g. we assume semilinear sets are given as finite unions of ultimately periodic sets of integers. A set $S \subseteq \mathbb{N}$ is *ultimately periodic* if there exist a threshold $t \in \mathbb{N}$ and a period $p \in \mathbb{N}$ such that for all $a, b \in \mathbb{N}$ with $a, b \geq t$ and $a \equiv b \mod p$, we have $a \in S$ iff $b \in S$. For complexity issues, all constants are assumed to be represented in binary. In that context, as a complexity parameter, we use $\#\mathsf{pred}_{\mathcal{A}}$, the number of predicates used on edges of $\mathcal{A}$.

3 Resolution of the parameterized reachability game problem

In this section, we study the complexity of the parameterized reachability game problem.

▶ **Theorem 5.** *The complexity of the parameterized reachability game problem is stated in Table 1.*

Note that the complexities for constraints given as (finite unions of) intervals are independent of values of endpoints used in the constraints. When constraints are given as semilinear sets, the complexity does depend on $\#\mathsf{pred}_{\mathcal{A}}$ as well as the size of the encodings of the semilinear sets.

The rest of this section is devoted to proving these complexity results. To do so, we start with defining a finite two-player game abstraction, the *knowledge game*, which precisely captures the partial-information aspect of our parameterized game model.

Table 1 Complexity of the parameterized reachability game problem.

		Deterministic arenas	Non-deterministic arenas
Constraints	Intervals	PTIME-complete	
	Finite unions of intervals	NP-complete	PSPACE-complete
	Semilinear sets	PSPACE-complete	

3.1 The knowledge game

From a parameterized reachability game, we construct a standard two-player turn-based game. We do not recall this notion here, and refer to [12, Chap. 2] for it.

▶ **Definition 6.** *Let $\mathcal{G} = (\mathcal{A}, t)$ be a parameterized game, with $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$. The* knowledge game *associated with $\mathcal{G}$ is the two-player turn-based reachability game $\mathcal{K}_\mathcal{G} = (V_E \cup V_A, \Delta_\mathcal{K}, F)$, between Eve and Adam, such that $V_E \subseteq V \times 2^{\mathbb{N}_{>0}}$ and $V_A \subseteq V_E \times \Sigma$ are Eve and Adam vertices, respectively; $\Delta_\mathcal{K} \subseteq (V_E \times V_A) \cup (V_A \times V_E)$ is the edge relation; and $F = V_E \cap \{(t, K) \mid K \subseteq \mathbb{N}_{>0}\}$ is the set of target vertices. They are defined inductively by*

- $\{(v, \mathbb{N}_{>0}) \mid v \in V\} \subseteq V_E$;
- $\forall (v, K) \in V_E$, $\forall a \in \Sigma$ *enabled at* v, $(v, K, a) \in V_A$ *and* $\big((v, K), (v, K, a)\big) \in \Delta_\mathcal{K}$;
- $\forall (v, K, a) \in V_A$, $\forall v' \in V$ *such that* $K \cap \nabla(v, a, v') \neq \emptyset$, $(v', K \cap \nabla(v, a, v')) \in V_E$ *and* $\big((v, K, a), (v', K \cap \nabla(v, a, v'))\big) \in \Delta_\mathcal{K}$;

A *strategy* for Eve in $\mathcal{K}_\mathcal{G}$ is a function $\lambda : (V_E \cdot V_A)^* \cdot V_E \to V_A$ compatible with $\Delta_\mathcal{K}$. We borrow standard notions of outcomes and winning strategies from the literature.

It is not hard to see that the game $\mathcal{K}_\mathcal{G}$ is finite. Indeed, one can show by induction that every Eve's vertex (v, K) (hence every Adam's vertex (v, K, a)) is such that K is an intersection of finitely many sets of the form $\nabla(v', a, v'')$ or $\mathbb{N}_{>0}$.

▶ **Example 7.** Figure 3 represents the knowledge game associated with the parameterized game from Example 2. Circle vertices belong to Eve, and rectangle ones to Adam. In this two-player game, Eve has a winning strategy from $(v_0, \mathbb{N}_{>0})$ to reach the doubly-circled target vertices.

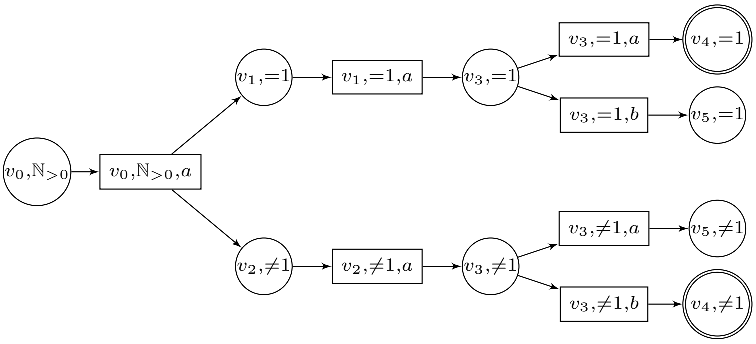


Figure 3 Knowledge game for the example of Figure 2.

We now investigate the size of $\mathcal{K}_\mathcal{G}$, that the number of its vertices and edges, w.r.t. the complexity measures we introduced for the parameterized game $\mathcal{G}$. Note that the size only might not reflect the complexity of *building* the knowledge game, in particular when constraints are given as semilinear predicates (one for instance needs to check emptiness of intersections of predicates); we discuss this further in the proof of Proposition 11.

▶ **Lemma 8.** *For $\mathcal{G} = (\mathcal{A}, t)$ a parameterized game with $\mathcal{A} = \langle V, \Sigma, \Delta \rangle$, the size of the associated knowledge game $\mathcal{K}_\mathcal{G}$ is polynomial in both $|V|$ and $|\Sigma|$, and*

1. *exponential in $\#\mathsf{pred}_\mathcal{A}$, for constraints defined by semilinear predicates;*
2. *exponential in $\#\mathsf{endpoints}_\mathcal{A}$, for constraints defined by finite unions of intervals; and*
3. *polynomial in $\#\mathsf{endpoints}_\mathcal{A}$, for constraints defined as intervals.*

Furthermore, the exponential blowup is unavoidable in the two first cases.

Proof. By definition, all pairs $(v, \mathbb{N}_{>0})$ for $v \in V$ belong to V_E representing that Eve has no initial knowledge of the number of her opponents. Further knowledge sets for vertices in $\mathcal{K}_\mathcal{G}$ are obtained by taking the intersection of existing knowledge sets with sets of the form $\nabla(v, a, v')$.

Therefore, when constraints in the arena are given by semilinear predicates, the number of knowledge sets is bounded by $2^{\#\mathsf{pred}_\mathcal{A}}$. Hence $|V_E| \le 2^{\#\mathsf{pred}_\mathcal{A}}|V|$ and $|V_A| \le 2^{\#\mathsf{pred}_\mathcal{A}}|V||\Sigma|$, yielding an overall exponential bound on $|\mathcal{K}_\mathcal{G}|$. Note that it is exponential in the number of predicates, but not in the size of their encodings.

When constraints are defined by finite unions of intervals, the number of knowledge sets is bounded by $3^{\#\mathsf{endpoints}_\mathcal{A}}$. Indeed, a finite union of intervals can be encoded by a word on the alphabet formed of the set of endpoints, with a repetition for singletons; for instance, if $E = \{2, 5, 8, 11, 17, 23, \infty\}$, writing a_i for the i-th letter of E, $[2, 8] \cup \{11\} \cup [17, \infty)$ is represented by the string $a_1a_3a_4a_4a_5a_7$. Hence $|V_E| \le 3^{\#\mathsf{endpoints}_\mathcal{A}}|V|$ and $|V_A| \le 3^{\#\mathsf{endpoints}_\mathcal{A}}|V||\Sigma|$, yielding an overall exponential bound on $|\mathcal{K}_\mathcal{G}|$. Note that it is exponential in the number of endpoints, but not in the size of their encodings.

Finally, when constraints are defined by intervals, a better upper bound can be obtained. All knowledge sets in $\mathcal{K}_\mathcal{G}$ are intervals whose endpoints appear in the constraints of $\mathcal{A}$. There can be at most $\#\mathsf{endpoints}_\mathcal{A}^2$ such intervals, so that $|V_E| \le \#\mathsf{endpoints}_\mathcal{A}^2|V|$ and $|V_A| \le \#\mathsf{endpoints}_\mathcal{A}^2|V||\Sigma|$, yielding an overall polynomial bound on $|\mathcal{K}_\mathcal{G}|$.

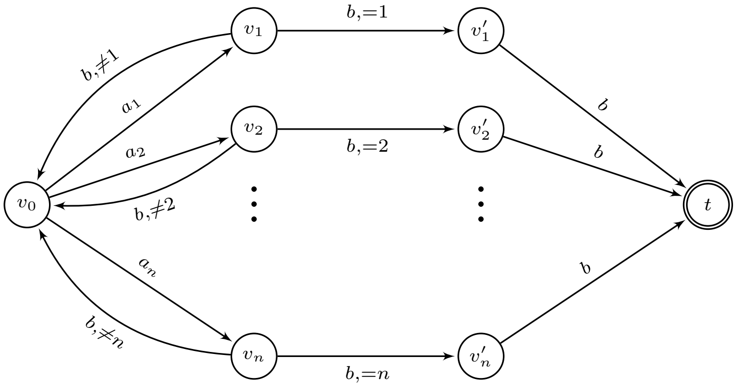


Figure 4 A deterministic game $\mathcal{G}_n$ ($n \in \mathbb{N}_{>0}$), whose size is polynomial in n and whose knowledge game is exponential in n.

The exponential upper bound is reached by the family $(\mathcal{G}_n)_{n\in\mathbb{N}_{>0}}$ of *deterministic* parameterized games depicted on Figure 4, and for which the constraints are unions of intervals (a particular case of semilinear predicates). Both the number of endpoints, and the number of predicates are linear in n. The associated knowledge game has vertices (v_0, K) for every non-empty subset K of $\{1, \ldots, n\}$. Indeed, intuitively, from vertex (v_0, K) in $\mathcal{K}_{\mathcal{G}_n}$, for any $k \in K$, the successor vertex in two steps by a_k and b, in case the number of opponents is not k, is the vertex $(v_0, K \setminus \{k\})$. Thus $|\mathcal{G}_n| \in O(n)$ and $|\mathcal{K}_{\mathcal{G}_n}| \in O(n2^n)$. ◀

We now state the correctness of the knowledge game construction:

▶ **Theorem 9.** *Eve has a winning strategy σ from v_0 in $\mathcal{G}$ if and only if she has a winning strategy λ from $(v_0, \mathbb{N}_{>0})$ in $\mathcal{K}_\mathcal{G}$.*

Proof sketch. There is a correspondence between histories in $\mathcal{G}$ and $\mathcal{K}_\mathcal{G}$. Every history $h = v_0 a_0 v_1 \cdots v_i$ in $\mathcal{G}$, can be lifted to the history $\kappa(h) = (v_0, K_0)(v_0, K_0, a_0)(v_1, K_1) \cdots (v_i, K_i)$ in $\mathcal{K}_\mathcal{G}$ where: $K_0 = \mathbb{N}_{>0}$, and for every $1 \leq j \leq i$, $K_j = K_{j-1} \cap \nabla(v_{j-1}, a_{j-1}, v_j)$. Note that $\kappa(h)$ is well-defined since, by definition of a history, K_i is not empty. Conversely, any history $H = (v_0, K_0)(v_0, K_0, a_0)(v_1, K_1) \cdots (v_i, K_i)$ in $\mathcal{K}_\mathcal{G}$ projects to $\iota(H) = v_0 a_0 v_1 \cdots v_i$ which is a history in $\mathcal{G}$. Moreover, for every $k \in K_i$, $\iota(H)$ is a k-history in $\mathcal{G}$. Using κ and ι, one can easily lift winning strategies from $\mathcal{G}$ to $\mathcal{K}_\mathcal{G}$ and, vice versa project winning strategies from $\mathcal{K}_\mathcal{G}$ to $\mathcal{G}$, to prove the desired equivalence. ◀

3.2 The simple case of intervals

▶ **Proposition 10.** *When constraints are intervals, the parameterized reachability game problem is* PTIME*-complete.*

When constraints are intervals only, the knowledge game is polynomial in the size of the parameterized arena (see Lemma 8) and it can be computed in polynomial time. Hence the parameterized reachability game problem is in PTIME. It is moreover complete for this class, since two-player reachability games are PTIME-hard (by straightforward reduction from the CIRCUIT-SAT problem). We thus obtain the above complexity result, independently of whether the arena is deterministic or not.

3.3 General PSPACE upper bound

▶ **Proposition 11.** *The parameterized reachability game problem is in* PSPACE *when constraints are given as finite unions of intervals or semilinear sets.*

Proof sketch. To prove this result, we rely on the knowledge game construction, which has been proven correct for the existence of winning strategies (see Theorem 9). Let $\mathcal{G} = (\mathcal{A}, t)$ be a parameterized reachability game, and v_0 be an initial vertex. We show that one can decide in polynomial space in the size of $\mathcal{G}$ whether Eve has a winning strategy from $(v_0, \mathbb{N}_{>0})$ in $\mathcal{K}_\mathcal{G}$.

For each vertex $(v, K) \in V_E$ of Eve in $\mathcal{K}_\mathcal{G}$, we define a reachability game $\mathcal{K}_\mathcal{G}[v, K]$, which is the restriction of $\mathcal{K}_\mathcal{G}$ to vertices (v', K, a) and (v', K') that are reachable from (v, K) via vertices with same knowlege set K only. Formally, $\mathcal{K}_\mathcal{G}[v, K]$ is the restriction of $\mathcal{K}_\mathcal{G}$ to the following sets of vertices, defined inductively:

$$\begin{cases} V_E^0 = \{(v, K)\} \\ V_A^i = \{(v', K, a) \mid v' \neq t \text{ and } (v', K) \in V_E^i \text{ and } \big((v', K), (v', K, a)\big) \in \Delta_\mathcal{G}\} \\ V_E^{i+1} = \{(v', K') \mid \exists (v'', K, a) \in V_A^i \text{ s.t. } \big((v'', K, a), (v', K')\big) \in \Delta_\mathcal{G}\} \end{cases}$$

Notice that in $\mathcal{K}_\mathcal{G}[v, K]$, all Adam vertices have knowledge set K. Also Eve vertices (v', K') with knowledge $K' \subsetneq K$ or with $v' = t$ have no successors: we refer to them as the *output vertices of* $\mathcal{K}_\mathcal{G}[v, K]$. We write $O^{[v,K]}$ for the set of such vertices.

The game $\mathcal{K}_\mathcal{G}[v, K]$ is polynomial in the size of $\mathcal{G}$. Indeed, there are at most $(|\Sigma| + 1)|V|$ many Eve or Adam vertices with second component exactly K and at most $|E||V|$ many Eve vertices with second component strictly smaller than K. When constraints are given as finite unions of intervals, this game can be computed in polynomial time in $\#\mathsf{endpoints}_\mathcal{A}$. For semilinear sets, $\mathcal{K}_\mathcal{G}[v, K]$ can be computed in polynomial space in the size of the encodings of the predicates as finite unions of ultimately periodic sets; in particular, if P is a semilinear predicate one needs to check whether $(P \cap K) \subsetneq K$ (to decide whether one obtains an output vertex of $\mathcal{K}_{\mathcal{G}}[v, K]$). Once constructed, $\mathcal{K}_{\mathcal{G}}[v, K]$ can be solved in polynomial time in $|\mathcal{G}|$ since this is a standard two-player turn-based reachability game. We use these games in sub-routines for solving the parameterized reachability game problem.

Using the subgames $\mathcal{K}_{\mathcal{G}}[v, K]$, we consider the following exponential-size tagged tree $\mathcal{T}$ defined inductively as follows: the root $\mathsf{n}_0 = (v_0, \mathbb{N}_{>0})$ is the initial vertex of $\mathcal{K}_{\mathcal{G}}$, and (v', K') is a child of (v, K) if $(v', K') \in O^{[v,K]}$ is an output vertex of $\mathcal{K}_{\mathcal{G}}[v, K]$. Our aim is to tag each node $\mathsf{n} = (v, K)$ of $\mathcal{T}$ with Win or Lose, to reflect whether Eve has a winning strategy from (v, K) in $\mathcal{K}_{\mathcal{G}}$. We define the following tagging function:

$$\mathsf{tag}((v,K)) = \begin{cases} \mathsf{Win} & \text{if } v = t \\ \mathsf{Win} & \text{if Eve has a winning strategy in } \mathcal{K}_{\mathcal{G}}[v,K] \text{ from } (v,K) \text{ to reach} \\ & \text{the set } \{\alpha \in O^{[v,K]} \mid \mathsf{tag}(\alpha) = \mathsf{Win}\} \\ \mathsf{Lose} & \text{otherwise.} \end{cases}$$

One can show the correctness of the tagging function: $\mathsf{tag}((v, K)) = \mathsf{Win}$ if and only if Eve has a winning strategy in $\mathcal{K}_{\mathcal{G}}$ from (v, K). Finally, the root of the tree can be tagged in polynomial space, by a a depth-first search algorithm on $\mathcal{T}$ (see Figure 5). The height of $\mathcal{T}$ is polynomially bounded, in $\#\mathsf{endpoints}_{\mathcal{A}}$ in the case of finite unions of intervals, and in $\#\mathsf{pred}_{\mathcal{A}}$ in the case of semilinear predicates. Once the tag of a node has been computed, its whole subtree can be forgotten. Therefore one can "reuse" polynomial space to repeatedly solve the games $\mathcal{K}_{\mathcal{G}}[v, K]$ for different v and K. In the DFS tagging, the size of the stack is at most the height of tree times the maximal number of successors of a vertex v in $\mathcal{G}$. Finally polynomial space is sufficient to store the knowledge of one node of the $\mathcal{T}$. ◀

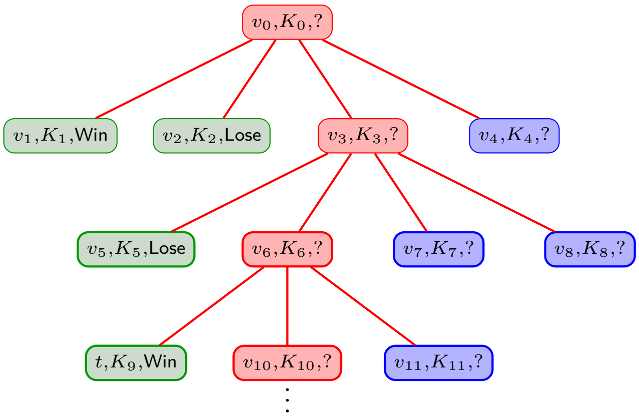


Figure 5 Illustration of the polynomial space DFS tagging algorithm: the Win/Lose tags of green nodes have already been computed (and their subtrees have been removed); the tags of red nodes are being computed (hence the label '?'); and the blue nodes are waiting to be processed (we also use label '?'). For instance, before tagging (v_6, K_6), one needs to first compute the tag of (v_{10}, K_{10}) (which is ongoing), then compute the tag of (v_{11}, K_{11}) (which is waiting).

3.4 An NP upper bound for deterministic arenas when constraints are finite unions of intervals

The previous PSPACE upper bound can be improved when the arena is deterministic and constraints are given by finite unions of intervals.

▶ **Proposition 12.** *The parameterized reachability game problem is in* NP*, when constraints are finite unions of intervals and when restricting to deterministic arenas.*

Proof sketch. Pick an arbitrary winning strategy σ for Eve, and consider the (labeled) tree $\mathcal{T}_\sigma$ it induces: nodes are histories, and the children of a node are the possible next histories (depending on the number of opponents). This tree is finite because σ is winning, and one can add to the node label the knowledge Eve has for the corresponding history. This tree satisfies the following properties: (i) along any path of $\mathcal{T}_\sigma$, the number of distinct knowledge sets is at most $\#\mathsf{endpoints}_{\mathcal{A}}$; and (ii) the knowledge at sibling nodes form a partition of the knowledge at their parent node. The second property has the following consequence. At each level of the tree, the knowledge of all nodes form a partition of $\mathbb{N}_{>0}$ using endpoints from the arena description, so that the number of nodes at each level is bounded by $\#\mathsf{endpoints}_{\mathcal{A}}$. Also, if a node has the same knowledge as its parent, it cannot have siblings. This allows to *compress* linear parts of the tree, and to tranform an arbitrary winning strategy into one whose tree is "small", *i.e.* polynomial in the size of the arena. ◀

3.5 Lower bounds

We prove all lower bounds mentioned in Table 1. We start with the PSPACE-hardness when constraints are finite unions of intervals and arenas are a priori non-deterministic.

▶ **Proposition 13.** *When constraints are finite unions of intervals, the parameterized reachability game problem is* PSPACE*-hard.*

Proof sketch. The proof is by reduction from QBF-SAT, which is known to be PSPACE-complete [16]. Let $\varphi = \exists x_1 \forall x_2 \exists x_3 \ldots \forall x_{2r} \cdot \big(C_1 \wedge C_2 \wedge \ldots \wedge C_m\big)$ be a quantified Boolean formula in prenex normal form, where for every $1 \leq h \leq m$, $C_h = \ell_{h,1} \vee \ell_{h,2} \vee \ell_{h,3}$, and for every $1 \leq j \leq 3$, $\ell_{h,j} \in \{x_i, \neg x_i \mid 1 \leq i \leq 2r\}$ are the literals. From φ, we construct an arena $\mathcal{A}_\varphi = \langle V, \Sigma, \Delta\rangle$ (see an illustrative example in Figure 6) as follows:

- $V = \{v_0, v_1, \ldots, v_{2r-1}, v_{2r}\} \cup \{v_{x_1}, v_{\bar{x}_1}, \ldots, v_{x_{2r}}, v_{\bar{x}_{2r}}\} \cup \{v_{C_1}, v_{C_2}, \ldots, v_{C_m}, v_{C_{m+1}}\} \cup \{\bot, \top\}$, where we identify v_{2r} with v_{C_1}, and $v_{C_{m+1}}$ with $\top$.
- $\Sigma = \{u, c\} \cup \bigcup_{1 \leq i \leq 2r} \{a_i, \bar{a}_i\}$
- For every $0 \leq s \leq r-1$, $1 \leq i \leq 2r$, $1 \leq h \leq m$ and $1 \leq j \leq 3$:
  1. $\Delta(v_{2s}, a_{2s+1}, \geq 1) = \{v_{x_{2s+1}}\}$ and $\Delta(v_{2s}, \bar{a}_{2s+1}, \geq 1) = \{v_{\bar{x}_{2s+1}}\}$
  2. $\Delta(v_{2s+1}, u, \geq 1) = \{v_{x_{2s+2}}, v_{\bar{x}_{2s+2}}\}$
  3. $\Delta(v_{x_i}, c, \neq 2i) = \{v_i\}$ and $\Delta(v_{x_i}, c, = 2i) = \{\top\}$
  4. $\Delta(v_{\bar{x}_i}, c, \neq 2i-1) = \{v_i\}$ and $\Delta(v_{\bar{x}_i}, c, = 2i-1) = \{\top\}$
  5. $\Delta(v_{C_h}, a_i, \neq 2i) = \{v_{C_{h+1}}\}$ if $\ell_{h,j} = x_i$; $\Delta(v_{C_h}, \bar{a}_i, \neq 2i-1) = \{v_{C_{h+1}}\}$ if $\ell_{h,j} = \neg x_i$

  To obtain a complete arena, all unspecified transitions lead to a sink state $\bot$.

From v_0, a first phase consists in choosing a valuation for the variables: Eve can choose the truth values of existentially quantified variables in vertices v_{2s} (with actions a_{2s+1} for true and $\bar{a}_{2s+1}$ for false), and her opponents resolve the non-determinism of action u (u stands for universal) to choose the truth values of universally quantified variables in vertices v_{2s-1}. Due to the constraints on the edges, the knowledge of Eve at v_{C_1} contains for every variable x_i, either $2i$ or $2i-1$ (and not both); where containing $2i$ (resp. $2i-1$) encodes the fact that x_i has been set to false by Eve or her opponents (resp. true).

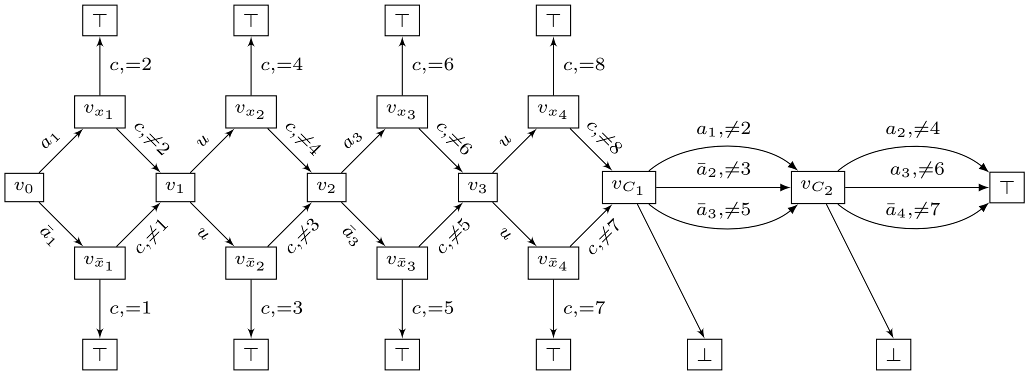


Figure 6 Reduction for formula $\varphi = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdot (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$. Knowledge of Eve at v_{C_1} contains for every variable x_i, either $2i$ or $2i{-}1$ (and not both); containing $2i$ (resp. $2i{-}1$) encodes that x_i has been set to false (resp. true).

From v_{C_1} a second phase starts where one checks whether the generated valuation makes all clauses in φ true. Sequentially, Eve chooses for every clause a literal that makes the clause true and these choices must be consistent with the first phase. To enforce this, plays with $2i{-}1$ and $2i$ opponents check the consistency of the assignment for variable x_i. For instance, if action a_i (encoding x_i set to true) against $2i{-}1$ opponents leads from v_{C_h} to $v_{C_{h+1}}$, this means that v_{x_i} was visited, hence that x_i was set to true. On the contrary, if v_{x_i} was not visited, hence x_i was set to false, then against $2i{-}1$ opponents, action a_i will lead to $\bot$. The role of $\bar{a}_i$ is dual; it encodes assigning false to x_i, and will be checked with plays against $2i$ opponents.

The above reduction ensures the following equivalence: Eve has a winning strategy in the parameterized game $\mathcal{G}_\varphi = (\mathcal{A}_\varphi, \top)$ if and only if φ is true. ◀

Note that the reduction can also be done with only three actions, which is the maximal number of enabled actions from any vertex. The reduction uses unions of intervals (due to $\neq i$ constraints). Finally the arena is non-deterministic at each vertex corresponding to universal quantifiers in φ. We extend this reduction in two ways to get rid of nondeterminism. First, instead of QBF-SAT, one can encode 3SAT (which is known to be NP-complete [8]) and obtain a deterministic parameterized game:

▶ **Corollary 14.** *When constraints are finite unions of intervals, and arenas are deterministic, the parameterized reachability game problem is* NP*-hard.*

Second, increasing the expressive power of predicates can encode universal quantifiers without nondeterminism:

▶ **Proposition 15.** *When constraints are semilinear sets and arenas are deterministic, the parameterized reachability game problem is* PSPACE*-hard.*

Proof sketch. We slightly modify the construction of the proof of Proposition 13 as shown on Figure 7. For every $1 \leq i \leq 2r$, p_i is the i-th prime number, and P_i the semilinear predicate "is a multiple of p_i".

Intuitively, at the end of the first phase, the truth value of variable x_i is witnessed by the fact that the set of possible number of opponents is a multiple of p_i if x_i is set to true (that is P_i is satisfied), and it is not a multiple of p_i if x_i is set to false (that is, $\neg P_i$ is satisfied). The rest of the proof is identical to that of Proposition 13. ◀

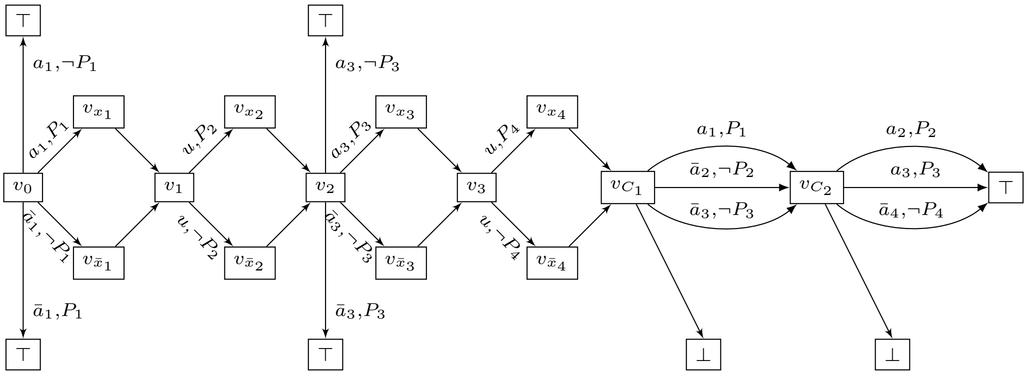


Figure 7 Reduction for formula $\varphi = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdot (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$. Predicate P_i is "divisible by i-th prime number".

4 Discussion: Beyond the number of players

Our model of parameterized game, with constraints on the number of opponents for Eve, is actually a simplification of a general concurrent game model, where the number of players is a parameter. This general model, motivated in introduction, is an extension of the multiplayer concurrent games of [2], where tuples of actions are replaced with languages.

▶ **Definition 16.** *A* language-based parameterized arena *is a tuple* $\mathcal{A}_{\text{L}} = \langle V, \Sigma, \Delta_{\text{L}} \rangle$ *where*

- V *is a finite set of vertices;*
- Σ *is a finite set of actions;*
- $\Delta_{\text{L}} : V \times \Sigma^{\geq 2} \to 2^V$ *is the transition function.*

The fact that Eve has at least one opponent explains the term $\Sigma^{\geq 2}$ in the transition function. We assume that for every $(v, v') \in V^2$, $\nabla_{\text{L}}(v, v') \overset{\text{def}}{=} \{w \in \Sigma^{\geq 2} \mid v' \in \Delta_{\text{L}}(v, w)\}$ is regular. Figure 1 in introduction provides an example of a language-based parameterized arena.

The game is then played as follows, when $k+1$ is the number of players, called Eve, Adam$_1$, …, Adam$_k$: from vertex v, each of the players select simultaneously and independently an action in Σ; concatenating all the letters (Eve first, and then all Adams' actions), it forms a word w; the next vertex of the game is then one of the vertices v' in $\Delta_{\text{L}}(v, w)$; the game then resumes from vertex v'. Strategies for Eve, and outcomes can be defined similarly to that of parameterized arenas in Section 2. The language-based parameterized game problem is then to decide whether Eve has a strategy that is winning against any number of opponents:

LANGUAGE-BASED PARAMETERIZED REACHABILITY GAME PROBLEM
Input: A language-based parameterized reachability game $\mathcal{G} = (\mathcal{A}, t)$ and a vertex v.
Question: Does Eve have a winning strategy from v in $\mathcal{G}$?

Language-based parameterized arenas generalize parameterized arenas: one can for instance replace rules of the form $v' \in \Delta(v, a, k)$ in a parameterized arena by $v' \in \Delta_{\text{L}}(v, a\Sigma^k)$ to construct a language-based parameterized arena, preserving the winning region for Eve. For our problem of existence of a winning strategy for Eve, the reduction in the other direction also holds:

▶ **Proposition 17.** *The language-based parameterized reachability game problem reduces in polynomial time to the parameterized reachability game (with semilinear predicates).*

Proof sketch. From a language-based parameterized arena, one can obtain an equivalent one (*i.e.* preserving the winning region for Eve) by first taking a left quotient of languages by any possible letter, and then projecting the obtained languages to lengths of words. Describing the reduction is simpler with the ∇ functions (and equivalent to using the Δ ones). We set $\nabla(v, a, v') = \{|u| \mid u \in a^{-1}\nabla_{\mathtt{L}}(v, v')\}$, where $a^{-1}\nabla_{\mathtt{L}}(v, v')$ is the left quotient by a of $\nabla_{\mathtt{L}}(v, v')$. Since $\nabla_{\mathtt{L}}(v, v')$ is regular, the set $\nabla(v, a, v')$ is semilinear [15]. Moreover, one can compute in polynomial time a representation for $\nabla(v, a, v')$ as a union of polynomially many ultimately periodic sets, with a polynomial encoding [7, 14]. Clearly enough this polynomial time reduction preserves the winning region for Eve. ◀

Thanks to Proposition 17, and using Propositions 11 and 15 we obtain the precise complexity of the language-based parameterized reachability game problem:

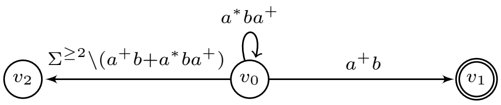
▶ **Theorem 18.** *The language-based parameterized reachability game problem is* PSPACE-*complete.*

5 Conclusion

In this paper, we introduce parameterized concurrent reachability games as a natural extension of the traditional concurrent games, where the number of players is unknown a priori. We consider different variants of a parameterized arena where the constraints on the number of opponents can be represented by intervals, finite unions of intervals, or semilinear sets. We have shown the existence of a uniform winning strategy for the first player to be PSPACE-complete in the general case, NP-complete when the arena is deterministic and the constraints are unions of intervals, and PTIME-complete when restricting to intervals only.

In this paper, we focused on reachability objectives. However the knowledge game approach also applies to more general objectives, like Büchi or parity, and even for quantitative objectives such as mean-payoff objectives. There is indeed a tight connection between strategies in the original game and strategies in the knowledge game, making the knowledge game abstraction correct for a variety of objectives. We plan to investigate complexity issues for objectives beyond reachability.

In future work, we also wish to investigate further this parameterized games model. In particular, it will be interesting to consider standard game theory concepts such as Nash equilibria. Also, to solve coordination problems, we will look for algorithms to synthesize strategies for all the players to achieve a global common goal. The figure below presents a simple coordination game, where we assume each player has a distinct identifier from 1 to some $n \in \mathbb{N}$, and their global objective is to reach the target vertex v_1.



If the players do not know beforehand the total number of players, but know their identifiers, a winning strategy profile is as follows: player i plays action a for the first $i-1$ steps, then plays b, and finally plays a for the remaining steps. Doing so, each player will in turn play action b, and when the last player does, the play reaches v_1. Synthesizing automatically winning profiles in such games is one of our long-term goals.

References

1 Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent Reachability Games. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 564–575. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743507.

2 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002. doi:10.1145/585265.585270.

3 Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, and Hugo Gimbert. Controlling a Population. In *Proceedings of the 28th International Conference on Concurrency Theory (CONCUR'17)*, volume 85 of *LIPIcs*, pages 12:1–12:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.12.

4 Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Playing with Probabilities in Reconfigurable Broadcast Networks. In *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'14)*, volume 8412 of *Lecture Notes in Computer Science*, pages 134–148. Springer, April 2014. doi:10.1007/978-3-642-54830-7_9.

5 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. doi:10.2200/S00658ED1V01Y201508DCT013.

6 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash Equilibria in Concurrent Games. *Logical Methods in Computer Science*, 11(2:9), 2015. doi:10.2168/LMCS-11(2:9)2015.

7 Marek Chrobak. Finite Automata and Unary Languages. *Theoretical Computer Science*, 47(3):149–158, 1986. doi:10.1016/0304-3975(86)90142-8.

8 Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC'71)*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

9 Giorgio Delzanno. Constraint-Based Verification of Parameterized Cache Coherence Protocols. *Formal Methods in System Design*, 23(3):257–301, 2003. doi:10.1023/A:1026276129010.

10 Javier Esparza. Keeping a Crowd Safe: On the Complexity of Parameterized Verification (Invited Talk). In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.1.

11 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational Synthesis. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.

12 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

13 Igor Konnov, Helmut Veith, and Josef Widder. What You Always Wanted to Know About Model Checking of Fault-Tolerant Distributed Algorithms. In *Proceedings of the 10th International Andrei Ershov Informatics Conference (PSI'15)*, volume 9609 of *Lecture Notes in Computer Science*, pages 6–21. Springer, 2015. doi:10.1007/978-3-319-41579-6_2.

14 Andrew Martinez. Efficient Computation of Regular Expressions from Unary NFAs. In *Proceedings of the 5th International Workshop on Descriptional Complexity of Formal Systems (DCFS'02)*, pages 174–187. Department of Computer Science, The University of Western Ontario, Canada, 2002.

15 Rohit Parikh. On Context-Free Languages. *Journal of the ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

16 Larry J. Stockmeyer and Albert R. Meyer. Word Problems Requiring Exponential Time (Preliminary Report). In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC'73)*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

17 Michael Ummels and Dominik Wojtczak. The Complexity of Nash Equilibria in Limit-Average Games. In *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2011. doi:10.1007/978-3-642-23217-6_32.

Expected Window Mean-Payoff

Benjamin Bordais
ENS Rennes, France

Shibashis Guha
Université libre de Bruxelles, Belgium

Jean-François Raskin
Université libre de Bruxelles, Belgium

Abstract

We study the expected value of the window mean-payoff measure in Markov decision processes (MDPs) and Markov chains (MCs). The window mean-payoff measure strengthens the classical mean-payoff measure by measuring the mean-payoff over a window of bounded length that slides along an infinite path. This measure ensures better stability properties than the classical mean-payoff. Window mean-payoff has been introduced previously for two-player zero-sum games. As in the case of games, we study several variants of this definition: the measure can be defined to be prefix-independent or not, and for a fixed window length or for a window length that is left parametric. For fixed window length, we provide polynomial time algorithms for the prefix-independent version for both MDPs and MCs. When the length is left parametric, the problem of computing the expected value on MDPs is as hard as computing the mean-payoff value in two-player zero-sum games, a problem for which it is not known if it can be solved in polynomial time. For the prefix-dependent version, surprisingly, the expected window mean-payoff value cannot be computed in polynomial time unless P=PSpace. For the parametric case and the prefix-dependent case, we manage to obtain algorithms with better complexities for MCs.

2012 ACM Subject Classification Mathematics of computing → Stochastic processes; Mathematics of computing → Probability and statistics

Keywords and phrases mean-payoff, Markov decision processes, synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.32

Related Version A full version of the paper is available at `https://arxiv.org/abs/1812.09298`.

Funding Work partially supported by the ARC project *Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond* (Fédération Wallonie-Bruxelles), and the EOS project *Verifying Learning Artificial Intelligence Systems* (F.R.S.-FNRS & FWO).

1 Introduction

Markov Decision processes (MDPs) are a classical model for decision-making in stochastic environments [16, 1]. Objectives in MDPs are formalized by functions that map infinite paths to values. Classical examples of such functions are the mean-payoff and the discounted sum [16]. The mean-payoff function does not guarantee local stability of the values along the path: if the mean-value of an infinite path is v, it is possible that for arbitrarily long infixes of the path, the mean-payoff of the infix is largely away from v. There have been several recent contributions [8, 4, 9, 5] that address this problem. Here, we study *window mean-payoff* objectives for MDPs; these objectives were first introduced in [8, 9] for two-player games.

In window mean-payoff [9], payoffs are considered over a local finite length window that slides along the path: the objective is to ensure that the mean-payoff always reaches a given threshold within the window length ℓ. This is a strengthening of classical mean-payoff: for all lengths ℓ, and all infinite sequences π of payoffs, if π satisfies the window mean-payoff objective for threshold v, then π has a mean-payoff of at least v. Interestingly, this additional stability property can always be met at the cost of a small degradation of mean-payoff performances in two-player games: whenever there exists a strategy with mean-payoff value

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 32; pp. 32:1–32:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Complexity, hardness and memory requirements for solving different window objectives for Markov decision processes and Markov chains (2-p stands for "two-player").

	MDP			**Markov chain**
	Complexity	**Memory**	**Hardness**	**Complexity**
WMP	polynomial (Thm. 5)	polynomial	2-p DirWMP (Thm. 9)	polynomial (Cor. of Thm. 5)
BWMP	UP ∩ coUP (Thm. 10)	memoryless	2-p Mean-payoff (Thm. 13)	polynomial[1] (Thm. 19)
DirWMP	exponential[2] (Thm. 15)	exponential	PSPACE (Thm. 16)	pseudopolynomial[3] (Thm. 21)

v then for every $\epsilon > 0$, there is a window length ℓ and a strategy that ensure that the window mean-payoff for threshold $v - \epsilon$ is eventually satisfied for windows of length ℓ (see Lemma 2(b) in [9]).

Here, we study how to maximize the expected value of the window mean-payoff function $\mathsf{f}^{\ell}_{\mathsf{DirWMP}}$ defined as follows: let $\pi : \mathbb{N} \to \mathbb{Z}$ be an infinite sequence of payoffs, then

$$\mathsf{f}^{\ell}_{\mathsf{DirWMP}}(\pi) = \sup\{\lambda \in \mathbb{R} \mid \forall i \in \mathbb{N} : \max_{1 \leq j \leq \ell} \frac{1}{j} \sum_{k=0}^{j-1} \pi(i+k) \geq \lambda\}$$

i.e., it returns the supremum of all thresholds that are enforced by the sequence of payoffs π for every window of length ℓ. As in [13], we study natural variants: (i) when the length of the window is fixed or it is left unspecified but needs to be bounded, and (ii) when the window property needs to be enforced from the beginning or not (leading to a prefix-independent variant.)

Main contributions. First, we provide an algorithm to compute the best expected value of $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ (prefix-independent version with fixed window length ℓ - noted WMP) with a time complexity polynomial in the size of the MDP and in ℓ (Theorem 5). As window mean-payoff objectives aim at strong stability over reasonable periods of time, it is natural to assume that ℓ is bounded polynomially by the size of the MDP, and so our algorithm is fully polynomial for those interesting cases. This complexity matches the complexity of computing the value of the function $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ for two-player games [9], and we provide a relative hardness result: deciding the existence of a winning strategy in a window mean-payoff game can be reduced in log-space to the problem of the expected value of $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ in an MDP (Theorem 9). Second, we consider the case in which the length ℓ in the measure $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ is not fixed but is required to be bounded (BWMP). We provide an algorithm in UP ∩ coUP (Theorem 10), and we show that providing a polynomial time solution for this case would give a polynomial time solution to the value problem in mean-payoff games (Theorem 13), a long-standing open problem [18]. Third, we consider the prefix-dependent version (DirWMP), i.e. the window property needs to hold directly from the beginning of the path. Surprisingly, this problem is expected to be harder: no polynomial time solution can exist unless P=PSPACE. Indeed, we show that this problem is PSPACE-HARD even if ℓ is given in unary (Theorem 16). We also provide an algorithm that executes in time that is polynomial in the size of the MDP, and in the largest weight appearing in the MDP, and exponential in the window length ℓ (Theorem 15). Finally, while our main results concentrate on MDPs, we also systematically provide results for the special case of Markov chains. An overview of our results is given in Table 1.

[1] independent of any window size
[2] exponential in window size and the number of bits to represent the weights on the edges
[3] pseudopolynomial in the number of bits to represent the weights on the edges

Related works. Window mean-payoff objectives were first introduced in [8] for two-player games, then for games with imperfect information in [13], and in combination with ω-regular constraints in [7]. Here, we consider them for MDPs instead of games. Still, we show that, for the prefix-independent version of the window objectives, inside an end-component of an MDP, the expected window mean-payoff value is closely related to the worst-case value of the associated zero-sum games (see Lemma 6 and Lemma 11). Stability issues of the mean-payoff measure triggered other works. First, in [4], MDPs with the objective to optimize the expected mean-payoff performance and stability are studied. Their notion of stability is related to statistical variance. The notion of stability offered by window mean-payoff objective studied here, is stronger. The techniques used to solve the two problems differ: in [4] they rely on solving sets of quadratic constraints, while our techniques rely on graph game algorithms and linear programming. Second, [5] introduces *window-stability objectives*. They are directly inspired from the window mean-payoff objective of [4] but contrary to window mean-payoff objectives, do not enjoy the inductive window property which is heavily used in our algorithms. Also, [5] considers games (2 players) and graphs (1 player) but not MDPs ($1\frac{1}{2}$ players).

MDP with classical mean-payoff objectives have been studied both for the threshold probability problem, in which the objective is to find a strategy that maximizes the probability that the mean-payoff is above a given threshold, and for the expectation problem that asks for a strategy that maximizes the expected value of the mean-payoff [16]. Combination of both types of constraints have been considered in [3]. The work of Brihaye et al. [6], appeared recently on arXiv, and was done independently of our work. The authors of [6] consider the threshold probability problem for window mean-payoff objectives in MDP: given a threshold $\lambda \in \mathbb{Q}$, and a window length ℓ, the problem asks to find a strategy that maximizes the probability of obtaining a window mean-payoff greater than or equal to λ. We study the expectation problem, and as for traditional mean-payoff objectives, the two problems are different and cannot be easily reduced to one another (see [3] and the discussion in the previous paragraph). Our work and their work are largely complementary. Some of the basic techniques employed in the two papers are however similar, e.g. for the prefix-independent objectives, both the works analyze maximal end components in related ways. Nevertheless, there are also important differences between the two works; e.g. we show that for the expected value, the prefix-dependent and the prefix-independent versions of the bounded window mean-payoff objective, lead to the same value; this is not the case for the threshold probability problem. We also show interesting connections between the fixed and the bounded case, for the expected value problem: the bounded case can be seen as the limit of the fixed case (Theorem 14), again this property does not hold for the threshold problem. Also, the algorithms for direct fixed window objective differ largely for the two problems. Though both the problems have been shown to be PSPACE-HARD, the expected value problem requires a more involved reduction. Finally, while we have shown how to solve the expected value problem for the special case of MCs for which we establish better complexity results, this is not considered in [6].

Structure of the paper. Sect. 2 introduces the necessary definitions and concepts. Sect. 3 defines the different variants of window mean-payoff objectives. Sect. 4 studies the prefix-independent variants while Sect. 5 covers the prefix-dependent variants. Algorithms and hardness results are given for all the problems. Finally, Sect. 6 considers the special case of MCs. We only provide sketches of the proofs here. Full proofs are given in [2].

2 Preliminaries

For $k \in \mathbb{N}$, we denote by $[k]_0$ and $[k]$ the set of natural numbers $\{0, \ldots, k\}$ and $\{1, \ldots, k\}$ respectively. Given a finite set A, a (rational) *probability distribution* over A is a function $\mathsf{Pr}\colon A \to [0,1] \cap \mathbb{Q}$ such that $\sum_{a \in A} \mathsf{Pr}(a) = 1$. We denote the set of probability distributions on A by $\mathcal{D}(A)$. The *support* of a probability distribution Pr on A is $\mathsf{Supp}(\mathsf{Pr}) = \{a \in A \mid \mathsf{Pr}(a) > 0\}$, and Pr is called *Dirac* if $|\mathsf{Supp}(\mathsf{Pr})| = 1$. An event is said to happen *almost surely* if it happens with probability 1.

Markov chain. A weighted *Markov chain* (MC, for short) is a tuple $\mathcal{M} = \langle S, E, s_{\mathsf{init}}, w, \mathbb{P}\rangle$, where S is a set of states, $s_{\mathsf{init}} \in S$ is an initial state, $E \subseteq S \times S$ is a set of edges, $w : E \to \mathbb{Q}$ maps edges to *weights* (or *payoff*), and $\mathbb{P} : S \to \mathcal{D}(E)$ assigns a probability distribution on the set $E(s)$ of outgoing edges from s. In the following, $\mathbb{P}(s,(s,s'))$ is denoted $\mathbb{P}(s,s')$, for all $s \in S$. The Markov chain $\mathcal{M}$ is *finite* if S is finite.

For $s \in S$, the set of *infinite paths* in $\mathcal{M}$ starting from s is $\mathsf{Paths}^{\mathcal{M}}(s) = \{\pi = s_0 s_1 \ldots \in S^\omega \mid s_0 = s, \forall n \in \mathbb{N}, \mathbb{P}(s_n, s_{n+1}) > 0\}$. The set of all the paths in $\mathcal{M}$ is $\mathsf{Paths}^{\mathcal{M}} = \bigcup_{s \in S} \mathsf{Paths}^{\mathcal{M}}(s)$. For a path $\pi = s_0 s_1 \ldots \in \mathsf{Paths}^{\mathcal{M}}$, by $\pi(i,l)$ we denote the sequence of $l+1$ states (or l edges) $s_i \ldots s_{i+l}$, and for simplicity, we denote $\pi(i,0)$ by $\pi(i)$. The infinite suffix of π starting in s_n is denoted by $\pi(n,\infty) \in \mathsf{Paths}^{\mathcal{M}}$. The set of *finite* paths starting from a state $s \in S$ is defined as $\mathsf{Fpaths}^{\mathcal{M}}(s) = \{\pi = s \ldots s' \in S^+ \mid \exists \overline{\pi} \in \mathsf{Paths}^{\mathcal{M}},\ \pi\overline{\pi} \in \mathsf{Paths}^{\mathcal{M}}(s)\}$ and $\mathsf{Fpaths}^{\mathcal{M}} = \bigcup_{s \in S} \mathsf{Fpaths}^{\mathcal{M}}(s)$. For $\pi = s \ldots s'$, we denote by $\mathsf{Last}(\pi)$, the last state s' in π.

Consider some measurable function $\mathsf{f} : \mathsf{Paths}^{\mathcal{M}}(s_{\mathsf{init}}) \to \mathbb{R}$ associating a value to each infinite path starting from s_{init}. For an interval $I \subseteq \mathbb{R}$, we denote by $\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, I)$ the set $\{\pi \in \mathsf{Paths}^{\mathcal{M}}(s_{\mathsf{init}}) \mid \mathsf{f}(\pi) \in I\}$, and for $r \in \mathbb{R}$, we denote by $\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, r)$ the set $\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, [r,r])$. Since the set of paths $\mathsf{Paths}^{\mathcal{M}}(s_{\mathsf{init}})$ forms a probability space, measured by a function Pr [17], and f is a random variable, we denote by $\mathbb{E}^{\mathcal{M}}_{s_{\mathsf{init}}}(\mathsf{f}) = \int_{x \in \mathbb{R}} \mathsf{Pr}(\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, x)) \cdot x$ the *expected value* of f over the set of paths starting from s_{init}.

The *bottom strongly connected components* (BSCCs for short) in a finite Markov chain $\mathcal{M}$ are the strongly connected components $\mathcal{B}$ from which it is impossible to exit, i.e. for all $s \in \mathcal{B}$ and $t \in \mathcal{M}$, if $\mathbb{P}(s,t) > 0$ then $t \in \mathcal{B}$. We denote by $\mathsf{BSCC}(\mathcal{M})$ the set of BSCCs of $\mathcal{M}$. Every infinite path eventually ends up in one of the BSCCs with probability 1. Considering $\Diamond$ and $\Box$ as the standard LTL *eventually* and *always* operators and that $\Diamond\Box\mathcal{B}$ denotes that eventually the path visits only states in $\mathcal{B}$ (see [1] for a formal definition), we formally state:

▶ **Proposition 1.** *For all $s \in S$, $\mathsf{Pr}(\pi \in \mathsf{Paths}^{\mathcal{M}}(s) \mid \exists \mathcal{B} \in \mathsf{BSCC}(\mathcal{M}),\ \pi \models \Diamond\Box\mathcal{B}) = 1$.*

Markov decision process. A finite weighted *Markov decision process* (MDP, for short) is a tuple $\Gamma = \langle S, E, Act, s_{\mathsf{init}}, w, \mathbb{P}\rangle$, where S is a finite set of states, $s_{\mathsf{init}} \in S$ is an initial state, Act is a finite set of actions, and $E \subseteq S \times Act \times S$ is a set of edges, the function $w : E \to \mathbb{Q}$ maps edges to *weights* (or *payoffs*), and $\mathbb{P} : S \times Act \to \mathcal{D}(E)$ is a function that assigns a probability distribution on the set $E(s,a)$ of outgoing edges from s if action $a \in Act$ is taken from s. Given $s \in S$ and $a \in Act$, we define $\mathsf{Post}(s,a) = \{s' \in S \mid \mathbb{P}(s,a)(s,s') > 0\}$. Then, for all states $s \in S$, we denote by $Act(s)$ the set of actions $\{a \in Act \mid \mathsf{Post}(s,a) \neq \emptyset\}$. We assume that, for all $s \in S$, we have $Act(s) \neq \emptyset$. In the following, we denote $\mathbb{P}(s,a)(s,s')$ by $\mathbb{P}(s,a,s')$.

A *strategy* in Γ is a function $\sigma : S^+ \to \mathcal{D}(Act)$ such that $\mathsf{Supp}(\sigma(s_0 \ldots s_n)) \subseteq Act(s_n)$, for all $s_0 \ldots s_n \in S^+$. We denote by $\mathsf{strat}(\Gamma)$ the set of strategies available in Γ. Once we fix a strategy σ in an MDP $\Gamma = \langle S, E, Act, s_{\mathsf{init}}, w, \mathbb{P}\rangle$, we obtain an MC $\Gamma^{[\sigma]}$ [1]. A strategy σ is *deterministic*, if for each $s_0 \ldots s_n \in S^+$, the distribution assigned by σ is Dirac, otherwise the strategy is *randomized*. We show that deterministic strategies suffice for playing optimally in all the problems considered here. For a sequence $\rho \in S^+$ of states, we also denote by

$\mathsf{Last}(\rho)$ the last state in ρ. Consider a measurable function f that associates a value to infinite paths in Markov chains. Then, we call $\sup_{\sigma\in\mathsf{strat}(\Gamma)} \mathbb{E}_{s_{\mathsf{init}}}^{\Gamma^{[\sigma]}}(\mathsf{f})$ the optimal expected value of f in Γ. In the sequel, when clear from the context, we denote $\sup_{\sigma\in\mathsf{strat}(\Gamma)} \mathbb{E}_{s_{\mathsf{init}}}^{\Gamma^{[\sigma]}}(\mathsf{f})$ by $\mathbb{E}_{s_{\mathsf{init}}}^{\Gamma}(\mathsf{f})$. A deterministic strategy can be encoded by a transition system $\langle Q, act, \delta, \iota\rangle$ where Q is a (possibly infinite) set of states, commonly called modes, $act : Q \times S \to Act$ selects an action such that, for all $q \in Q$ and $s \in S$, $act(q, s) \in Act(s)$, $\delta : Q \times S \to Q$ is a mode update function and $\iota : S \to Q$ selects an initial mode for each state $s \in S$. The amount of memory used by such a strategy is defined to be $|Q|$. A strategy is said to be *memoryless* if $|Q| = 1$, that is, the choice of action only depends on the current state where the choice is made. Formally, a strategy is memoryless if for all finite sequences of states ρ_1 and ρ_2 in S^+ such that $\mathsf{Last}(\rho_1) = \mathsf{Last}(\rho_2)$, we have $\sigma(\rho_1) = \sigma(\rho_2)$. A strategy is called *finite memory* if Q is finite. Note that the state space of $\Gamma^{[\sigma]}$ is $S \times Q$. For a sequence π of states in $\Gamma^{[\sigma]}$, we denote by $\mathsf{proj}(\pi)_{|S}$ the corresponding sequence of states in the MDP Γ.

An *end-component* (EC, for short) $M = (T, A)$ with $T \subseteq S$, and $A : T \to 2^{Act}$ is a *sub-MDP* of Γ (for all $s \in T$, we have $A(s) \subseteq Act(s)$, and for all $a \in A(s)$, we have $\mathsf{Post}(s, a) \subseteq T$) that is strongly connected. A *maximal EC* (MEC, for short) is an EC that is not included in any other EC. We denote by $\mathsf{MEC}(\Gamma)$ the set of all maximal end components of Γ. Any infinite path will eventually end up in one maximal end component almost surely, whatever strategy is considered. This is stated in the following proposition:

▶ **Proposition 2** ([11]). *In an MDP Γ, for each strategy $\sigma \in \mathsf{strat}(\Gamma)$, for every state $s \in S$, and mode $q \in Q$, we have:* $\mathsf{Pr}(\pi \in \mathsf{Paths}^{\Gamma^{[\sigma]}}(s, q) \mid \exists M = (T, A) \in \mathsf{MEC}(\Gamma), \mathsf{proj}(\pi)_{|S} \models \Diamond\Box T) = 1$.

Weighted two-player games. An MDP can also be considered to have the semantics of a two-player turn-based game (denoted 2P) played for infinitely many rounds while ignoring the probabilities. Every 2P we consider here can be played optimally with deterministic strategies, therefore we restrict ourselves to deterministic strategies for both players. The first round starts from s_{init}. In each round, Player 1 chooses an action $a \in Act(s)$ from a state s while Player 2 chooses a state $s' \in \mathsf{Post}(s, a)$. We denote by $G_\Gamma = \langle S, E, Act, s_{\mathsf{init}}, w\rangle$ the two-player game that is obtained from an MDP $\Gamma = \langle S, E, Act, s_{\mathsf{init}}, w, \mathbb{P}\rangle$.

For a 2P, Player 1 thus chooses among the deterministic strategies available in MDPs. A strategy of Player 2 is a function $\mu : S^+{\cdot}Act \to S$, with the restriction that if $\mu(s_0 s_1 \dots s_n{\cdot}a) = s$ then $\mathbb{P}(s_n, a, s) > 0$. The set of deterministic strategies for Player 1 and Player 2 are denoted by $\mathsf{strat}_1(G)$ and $\mathsf{strat}_2(G)$ respectively. In a two-player game there is no randomness: Given two strategies $\sigma_1 \in \mathsf{strat}_1(G)$ and $\sigma_2 \in \mathsf{strat}_2(G)$, we denote by $\pi_{(G,s,\sigma_1,\sigma_2)}$ the unique path that occurs in 2P G under strategies σ_1 and σ_2 from state s. Then, for a function f that associates a value to each infinite path, we denote by $V_s^{\mathsf{f}}(G)$ the value $\sup\limits_{\sigma_1\in\mathsf{strat}_1(G)} \inf\limits_{\sigma_2\in\mathsf{strat}_2(G)} \mathsf{f}(\pi_{(G,s,\sigma_1,\sigma_2)})$. The definitions of the memories of strategies also apply to two-player games.

In the following, in MCs, MDPs and in 2Ps, w.l.o.g. we consider only non-negative integer weights[4]. We denote by W the maximum weight appearing on the edges for MCs, MDPs and 2Ps. We denote the size of an MC $\mathcal{M}$, MDP Γ and 2P G by $|\mathcal{M}|$, $|\Gamma|$ and $|G|$ respectively. This size is equal to $|S| + |E|$.

[4] For weights belonging to $\mathbb{Q}$, we can multiply them with the LCM d of their denominators to obtain integer weights. Among the resultant set of integer weights, if the minimum integer weight κ is negative, then we add -κ to the weight of each edge so that the resultant weights are natural numbers. For a function f if the expected value was originally x, then the new expected value is $d \cdot x - \kappa$.

3 Window Mean-Payoff Value

Let $\mathcal{M} = \langle S, E, s_{\mathsf{init}}, w, \mathbb{P}\rangle$ be a finite MC. Let $\rho = s_0 \dots s_n \in \mathsf{Fpaths}^{\mathcal{M}}(s)$, we define $\mathsf{MP} : \mathsf{Fpaths}^{\mathcal{M}} \to \mathbb{Q}$ as: $\mathsf{MP}(\rho) = \frac{1}{n}\sum_{i=0}^{n} w(s_i, s_{i+1})$, where $n = |\rho| > 0$, the number of edges in ρ. For $\pi = s_0 \dots \in \mathsf{Paths}^{\mathcal{M}}$, the *mean-payoff* function $\mathsf{f_{Mean}} : \mathsf{Paths}^{\mathcal{M}} \to \mathbb{R}$ is defined as

$$\mathsf{f_{Mean}}(\pi) = \liminf_{n\to\infty} \mathsf{MP}(s_0 \dots s_n) \tag{1}$$

We now define several variants of *window mean-payoff value functions*. For $\pi = s_0 s_1 \dots s_n \dots \in \mathsf{Paths}^{\mathcal{M}}$, a window size ℓ, and a position i, the window mean-payoff value of π in position i over length ℓ is defined by $\mathsf{WMP}^{\ell}(\pi(i,\infty)) = \max_{k\in[\ell]} \mathsf{MP}(\pi(i,k))$, i.e. it is the maximal value of the mean-payoff of an infix of π that starts at position i and with a size at most ℓ. For a threshold λ such that $\mathsf{WMP}^{\ell}(\pi(i,\infty)) \geq \lambda$, we say that the window mean-payoff value over length ℓ is at least λ at position i. We define the *fixed window mean-payoff function* $\mathsf{f}^{\ell}_{\mathsf{WMP}} : \mathsf{Paths}^{\mathcal{M}} \to \mathbb{R}$ such that, for every path $\pi = s_0 s_1 \dots s_n \dots \in \mathsf{Paths}^{\mathcal{M}}$:

$$\mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi) = \sup\{\lambda \in \mathbb{R} \mid \exists k \in \mathbb{N},\ \forall i \geq k : \mathsf{WMP}^{\ell}(\pi(i,\infty)) \geq \lambda\} \tag{2}$$

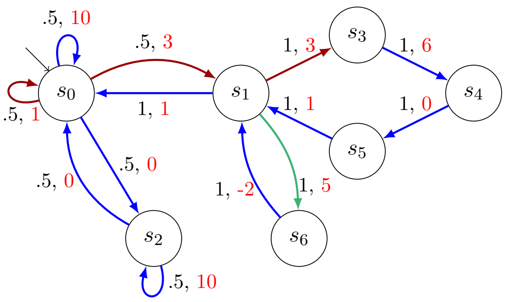


Figure 1 In the MEC M with initial state s_0, the expected value of $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ (resp. $\mathsf{f_{BWMP}}$) is the maximum of the value of the two-player game with the direct fixed window mean-payoff (resp. classical mean-payoff) objective obtained over all states. Also, for $\ell = 3$, we have $\mathbb{E}^{M}_{s_0}(\mathsf{f}^{\ell}_{\mathsf{WMP}}) < \mathbb{E}^{M}_{s_0}(\mathsf{f_{BWMP}}) < \mathbb{E}^{M}_{s_0}(\mathsf{f_{Mean}})$.

The value $\mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi)$ corresponds to the supremum over all thresholds λ where for every such λ, there exists a position k such that for all positions $i \geq k$, the window mean-payoff value over length ℓ is at least λ. We note some properties of the function $\mathsf{f}^{\ell}_{\mathsf{WMP}}$. First, it is prefix-independent, that is, for every path $\pi \in \mathsf{Paths}^{\mathcal{M}}$, for all $n \geq 1$, we have $\mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi) = \mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi(n,\infty))$. Second, it is a strengthening of the classical mean-payoff function: for all paths π, we have that $\mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi) \leq \mathsf{f_{Mean}}(\pi)$. And finally, $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ imposes strong stability properties: if $\mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi) \geq \lambda$, then from some point on in π, it is always the case that the observed mean-payoff from position i gets larger than λ within position $i + \ell$. This stability property is not enforced by classical mean-payoff function for which infixes of arbitrary lengths can have arbitrary low mean-payoffs.

Then, we define the *bounded window mean-payoff function* $\mathsf{f_{BWMP}} : \mathsf{Paths}^{\mathcal{M}} \to \mathbb{R}$ such that, for every path $\pi = s_0 \dots \in \mathsf{Paths}^{\mathcal{M}}$:

$$\mathsf{f_{BWMP}}(\pi) = \sup\{\lambda \in \mathbb{R} \mid \exists \ell, k \geq 1, \forall i \geq k : \mathsf{WMP}^{\ell}(\pi(i,\infty)) \geq \lambda\} \tag{3}$$

Here, the length of the window is not fixed but it needs to be bounded.

Now, we define the *direct fixed window mean-payoff function* $\mathsf{f}^{\ell}_{\mathsf{DirWMP}} : \mathsf{Paths}^{\mathcal{M}} \to \mathbb{R}$ such that, for every path $\pi = s_0 \ldots \in \mathsf{Paths}^{\mathcal{M}}$:

$$\mathsf{f}^{\ell}_{\mathsf{DirWMP}}(\pi) = \sup\{\lambda \in \mathbb{R} \mid \forall i \geq 0 : \mathsf{WMP}^{\ell}(\pi(i,\infty)) \geq \lambda\} \tag{4}$$

Here the window property must hold from the beginning of the path and so it is not prefix-independent. For every path $\pi \in \mathsf{Paths}^{\mathcal{M}}$, we have $\mathsf{f}^{\ell}_{\mathsf{DirWMP}}(\pi) \leq \mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi)$. Finally, we define the *direct bounded window mean-payoff function* $\mathsf{f}_{\mathsf{DirBWMP}} : \mathsf{Paths}^{\mathcal{M}} \to \mathbb{R}$ such that, for every path $\pi = s_0 \ldots \in \mathsf{Paths}^{\mathcal{M}}$:

$$\mathsf{f}_{\mathsf{DirBWMP}}(\pi) = \sup\{\lambda \in \mathbb{R} \mid \exists \ell \geq 1, \forall i \geq 0 : \mathsf{WMP}^{\ell}(\pi(i,\infty)) \geq \lambda\} \tag{5}$$

i.e., variant where the length of the window is not fixed.

The following proposition relates some of the variants defined above in a Markov chain $\mathcal{M}$.

▶ **Proposition 3.** *Let $\pi \in \mathsf{Paths}^{\mathcal{M}}$. Then, we have: $\sup_{\ell \geq 1} \mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi) = \mathsf{f}_{\mathsf{BWMP}}(\pi) \leq \mathsf{f}_{\mathsf{Mean}}(\pi)$.*

▶ **Example 4.** Consider the example in Figure 1 where the MDP Γ is a single MEC. The probabilities appear in black and the weights in red. The strategy that chooses the blue action in s_0 and in s_2 maximizes the expected value of the classical mean-payoff function $\mathsf{f}_{\mathsf{Mean}}$ in Γ from s_0. The expected value of this strategy is 5. However, clearly, while playing this strategy, we run the risk of having a mean-payoff of 0 for arbitrarily long period (while looping between s_0 and s_2). So it may not be the best strategy if we aim at some stability property in the mean-payoff. In this example, the strategy that maximizes the expected value of $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ for $\ell = 3$, is the strategy that plays the brown action in state s_0 and then alternates between the brown and green action in s_1.

4 Algorithms and Hardness for Prefix-independent Objectives

The fixed window mean-payoff function for length ℓ can be solved in time that is polynomial in the size of the MDP and in ℓ:

▶ **Theorem 5.** *Given an MDP Γ with maximum weight W, a window length ℓ and a threshold $\lambda \in \mathbb{Q}$, whether $\mathbb{E}^{\Gamma}_{s_{init}}(\mathsf{f}^{\ell}_{\mathsf{WMP}}) \geq \lambda$ can be decided in $O(\mathsf{poly}(|\Gamma|, \ell, \log_2 W))$ time and deterministic polynomial memory strategies suffice to play optimally.*

To establish this result, we first study the case of a single MEC $M = (T, A)$. By Proposition 2, for every strategy σ, each path of $\Gamma^{[\sigma]}$ will almost surely end up in an MEC. Since $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ is prefix-independent, the value of a path only depends on its behavior in the MEC in which it ends up. Since M is strongly connected (as it is an MEC), for every $s, s' \in T$, there exists a strategy $\sigma_{(s,s')} \in \mathsf{strat}(M)$ such that every path starting from s reaches s' almost surely, in the Markov chain $M^{[\sigma_{(s,s')}]}$. Therefore, for all $s, s' \in T$, we have $\mathbb{E}^{\Gamma}_{s}(\mathsf{f}^{\ell}_{\mathsf{WMP}}) = \mathbb{E}^{\Gamma}_{s'}(\mathsf{f}^{\ell}_{\mathsf{WMP}})$, i.e. the optimal expected value is the same from all states in the MEC. We denote by λ^{ℓ}_{M} this optimal value. Now, the following lemma interestingly relates λ^{ℓ}_{M} to the maximum over all states s of the optimal *adversarial value* from s (which is the value of $V_s^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}$), that is when the stochastic behavior in M is replaced by an adversary:

▶ **Lemma 6.** *Let $M = (T, A)$ be an MEC that is also an MDP. Then $\lambda^{\ell}_{M} = \max\limits_{s \in T} V_s^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}(G_M)$.*

Proof sketch. Let $v \in T$ be a state that maximizes the value of the 2P that is, $V_v^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}(G_M) = \max_{s \in T} V_s^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}(G_M)$. When a strategy σ is fixed in M, using classical probability arguments (Borel-Cantelli) every possible finite sequence of states (with respect to the strategy σ) is visited infinitely often almost surely. In particular, the worst sequence of states in terms of maximizing the fixed window mean-payoff (that is the sequence that Player 2 chooses in the two-player game G_M) is visited infinitely often almost surely. Hence, the expected window mean-payoff in the MEC M is at most the value of the 2P G_M from v, that is $V_v^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}(G_M)$.

Now, consider a strategy in the MEC M that consists in reaching v and then playing according to an optimal deterministic strategy of Player 1 in the two-player game G_M from v. Then, every path in M consistent with that strategy has a window mean-payoff of at least $V_v^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}(G_M)$. Thus the expected value of the window mean-payoff is at least $V_v^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}(G_M)$. ◀

To solve the two-player game, we rely on the following result from [9]:

▶ **Theorem 7.** *Given a* 2P *with maximum weight W, a window length ℓ, and a threshold $\lambda \in \mathbb{Q}$, in a two-player window mean-payoff game, for both the fixed window and the direct fixed window mean-payoff objectives, it can be decided in* $O(\mathsf{poly}(|G|, \ell, \log_2 W))$ *time if Player* 1 *has a winning strategy. For both players, an optimal strategy may need memory that is linear in $|G|$ and ℓ and such strategies can be constructed in time* $O(\mathsf{poly}(|G|, \ell, \log_2 W))$*, and deterministic strategies suffice to play optimally.*

▶ **Example 8.** Consider again the example of Figure 1. Lemma 6 tells us that we need to compute the two-player game value of the direct fixed window objective for $\ell = 3$ at each state of the MEC. We can check that this value is equal to 2 for all states but s_0 and s_2 in which the game values are equal to 1 and 2/3 respectively. Now, to obtain the best expected value for $\mathsf{f}^{\ell}_{\mathsf{WMP}}$ with $\ell = 3$ from s_0, we must play a strategy that first reaches almost surely any state $s \notin \{s_0, s_2\}$ and then switches to an optimal strategy for the two-player game from s.

As we know how to deal with an MEC, we now consider the general case.

Proof sketch of Theorem 5. Our algorithm for solving the general case proceeds as follows: (i) it decomposes Γ into MECs, (ii) for each MEC M, it computes the value λ^{ℓ}_{M} as described in Lemma 6, (iii) it constructs a new MDP Γ^{MEC} that is identical to Γ except that every MEC $M \in \mathsf{MEC}(\Gamma)$ is now compacted into a single state s_M, the transition relation is defined accordingly to mimic the transition relation of Γ over its MECs, the value of each transition that self-loops on s_M is assigned the value λ^{ℓ}_{M}, as computed in point (ii), and the other transitions have the same value as in Γ, (iv) it computes the optimal expected (classical) mean-payoff value for the new MDP Γ^{MEC}. It should be clear that the optimal expected mean-payoff of Γ^{MEC} is equal to the optimal expected window mean-payoff value in Γ.

Now we analyze the complexity of this algorithm. The MEC decomposition of Γ of step (i) can be done in quadratic time [10] in the size of Γ yielding at most $|S|$ MECs. By Theorem 7, given a threshold λ, for every MEC M and for each state s in M, it can be decided in time $O(\mathsf{poly}(|M| \cdot \ell \cdot \log_2 W))$ whether Player 1 has a winning strategy for the direct fixed window mean-payoff game from s. To find the maximal expected window mean-payoff in M, we do a binary search over a set $\Lambda = \{\frac{p}{q} \mid q \in [\ell],\ p \in [q \cdot W]_0\}$ with $|\Lambda| = O(W \cdot \ell^2)$ different possible values of λ and decide the two-player game starting from each state in M for each such λ. Furthermore, the construction of Γ^{MEC} can be done in time $O(|\Gamma|)$. Finally, the maximal expected value of the (classical) mean-payoff in Γ^{MEC} can be computed in polynomial time (see e.g. [16]) using linear programming. Thus all the steps can be done in time $O(\mathsf{poly}(|\Gamma|, \ell, \log_2 W))$.

We construct the optimal strategy σ from steps $(i) - (iii)$ by combining them with a deterministic memoryless strategy that optimizes the expected value of the (classical) mean-payoff in Γ^{MEC} (step (iv)). When this memoryless strategy prescribes to stay in an MEC M, we apply inside M the strategy defined in the proof of Lemma 6. The memory used by the strategy σ is polynomial in $|\Gamma|$ and ℓ as announced. ◀

The algorithm above relies on solving two-player games for the direct fixed window mean-payoff objective. We next show that this step cannot be improved without improving the algorithms for those games. Indeed, the following relative hardness result holds: solving the two-player game for the direct fixed window mean-payoff objective can be reduced in log-space to computing the expected value of the fixed window mean-payoff function.

▶ **Theorem 9.** *Given a* 2P *G with an initial state s_{init} and a window length ℓ, we can construct in log-space an MDP Γ_G with an initial state s'_{init} such that $\mathbb{E}^{\Gamma_G}_{s'_{\mathit{init}}}(\mathsf{f}^{\ell}_{\mathsf{WMP}}) = V^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}_{s_{\mathit{init}}}(G)$.*

Proof sketch. Consider a weighted two-player game $G = \langle S, E, \mathit{Act}, s_{\mathsf{init}}, w\rangle$. We construct an MDP Γ from G such that $\mathbb{E}^{\Gamma}_{s_{\mathsf{init}}}(\mathsf{f}^{\ell}_{\mathsf{WMP}}) = V^{\mathsf{f}^{\ell}_{\mathsf{DirWMP}}}_{s_{\mathsf{init}}}(G)$.

We first construct another game $G^{reset} = \langle S, E', \mathit{Act}, s_{\mathsf{init}}, w'\rangle$ from G where $E' = E \cup \{(s, a, s_{\mathsf{init}}) \mid (s, a, s_{\mathsf{init}}) \notin E, s \in S \setminus \{s_{\mathsf{init}}\}$ and $a \in \mathit{Act}(s)\}$ and $w'(e) = w(e)$ for $e \in E$ and $w'(e) = (W + 1) \cdot \ell$ for $e \in E' \setminus E$. Note that the game graph of G^{reset} is strongly connected. In the game G^{reset}, since Player 2 may "reset" the game at any time by taking an edge to s_{init}, the maximum of the value over all starting states of the two-player game is achieved at the state s_{init}. Moreover, the weight on these new edges being high enough, it is not in the interest of Player 2 to take one of them more than once. It follows that the values of the two-player game, starting from s_{init}, for the direct fixed window objective are the same in G and G^{reset}.

Now considering G^{reset} as an MDP $\Gamma = \langle S, E', \mathit{Act}, s_{\mathsf{init}}, w', \mathbb{P}\rangle$, such that for all $e \in E'$, we have $\mathbb{P}(e) > 0$, we note that Γ is actually an MEC. The result follows from Lemma 6. ◀

We now consider the prefix-independent version of the bounded window mean-payoff objective. For that case, we provide a UP $\cap$ coUP solution.

▶ **Theorem 10.** *Given an MDP Γ and a threshold $\lambda \in \mathbb{Q}$, deciding whether $\mathbb{E}^{\Gamma}_{s_{\mathit{init}}}(\mathsf{f}_{\mathsf{BWMP}}) \geq \lambda$ is in* UP $\cap$ coUP *and deterministic memoryless strategies suffice to play optimally.*

Since $\mathsf{f}_{\mathsf{BWMP}}$ is prefix-independent, similar to the fixed case, we first consider a single MEC M. All the states in the MEC M have the same value λ_M, and surprisingly, this value is the maximum over all states s of M of the optimal adversarial value from s (i.e. when the stochastic behavior is replaced by an adversary), for the *classical mean-payoff value* $V^{\mathsf{f}_{\mathsf{Mean}}}_s(G_M)$:

▶ **Lemma 11.** *Let $M = (T, A)$ be an MEC that is also an MDP. Then $\lambda_M = \max\limits_{s \in T} V^{\mathsf{f}_{\mathsf{Mean}}}_s(G_M)$.*

Proof sketch. Let v be a state such that $V^{\mathsf{f}_{\mathsf{Mean}}}_v(G_M) = \max_{s \in T} V^{\mathsf{f}_{\mathsf{Mean}}}_s(G_M)$.

Consider an optimal strategy σ_2 for Player 2 (that can be chosen among deterministic memoryless strategies). Now, let $\sigma \in \mathsf{strat}(M)$ (note that σ may be a randomised strategy), $\ell \geq 1$ and s be a state in the Markov chain $M^{[\sigma]}$. Any path π compatible with strategies σ and σ_2 must ensure $V^{\mathsf{f}_{\mathsf{Mean}}}_v(G_M) \geq \mathsf{f}_{\mathsf{Mean}}(\pi) \geq \mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi)$ (the last inequality is given by Proposition 3). Hence, in the MC $M^{[\sigma]}$, there is a non-zero probability to reach a sequence of states whose window mean-payoff is below $V^{\mathsf{f}_{\mathsf{Mean}}}_v(G_M)$ from s. This is true for every state s in $M^{[\sigma]}$. It follows that for every path $\pi \in \mathsf{Paths}^{M^{[\sigma]}}$, almost surely a sequence of states whose window mean-payoff is at most $V_v^{\mathsf{f}_{\mathsf{Mean}}}(G_M)$ is visited infinitely often. Therefore, the fixed window mean-payoff for length ℓ of a path in $M^{[\sigma]}$ is almost surely at most $V_v^{\mathsf{f}_{\mathsf{Mean}}}(G_M)$. This is true for every $\ell \geq 1$. Hence, by Proposition 3, we have that the bounded window mean-payoff of a path in $M^{[\sigma]}$ is at most $V_v^{\mathsf{f}_{\mathsf{Mean}}}(G_M)$ almost surely. Thus, $\mathbb{E}^{M[\sigma]}(\mathsf{f}_{\mathsf{BWMP}}) \leq V_v^{\mathsf{f}_{\mathsf{Mean}}}(G_M)$. This holds for every strategy $\sigma \in \mathsf{strat}(M)$. Therefore, $\lambda_M \leq V_v^{\mathsf{f}_{\mathsf{Mean}}}(G_M)$.

Now, consider an optimal strategy $\sigma_1 \in \mathsf{strat}_1(G_M)$ for Player 1 in G_M. Let ρ be a cycle of mean-payoff m that is minimal among all the cycles compatible with σ_1. Note that $V_v^{\mathsf{f}_{\mathsf{Mean}}}(G_M)$ equals m. Every path $\pi \in \mathsf{Paths}^{M^{[\sigma_1]}}$ has a bounded window mean-payoff of at least m since every cycle appearing in π has a mean-payoff of at least m, and for every ε, there exists $\ell > 0$ such that a direct fixed window mean-payoff of $m - \varepsilon$ can be ensured for every window of length ℓ along π. Thus $\lambda_M \geq m = \max_{s \in T} V_s^{\mathsf{f}_{\mathsf{Mean}}}(G_M)$ and hence the result. ◀

▶ **Example 12.** Consider again the example of Figure 1. Lemma 11 tells us that we need to compute the two-player game value of the classical mean-payoff objective $\mathsf{f}_{\mathsf{Mean}}$ at each state of the MEC. We can check that this value is equal to 2.5 for all states (by taking the brown action from s_1) but s_0 and s_2 at which the game value is equal to 1 . Now, to obtain the best expected value for $\mathsf{f}_{\mathsf{BWMP}}$, we must play a strategy that first reaches almost surely, from s_0, any other state $s \notin \{s_0, s_2\}$, (e.g. always play brown) and then switches to the optimal strategy for the two-player game from s for the classical mean-payoff objective.

We can now prove our main theorem for the bounded window mean-payoff objective.

Proof sketch for Theorem 10. The algorithm for this case follows exactly the algorithm in four steps (i), (ii), (iii), and (iv) of the algorithm for the proof of Theorem 5, with the difference, that step (ii) computes λ_M instead of λ_M^ℓ, and we use Lemma 11 to this end. The complexity of the algorithm is no more polynomial but in UP ∩ coUP because step (ii) requires solving a mean-payoff game [18, 14]. To construct an optimal strategy, we follow the same recipe as in the proof of Theorem 5. In this case, the strategies are deterministic and memoryless (mean-payoff games can be played optimally with memoryless strategies) and so deterministic memoryless strategies are sufficient to obtain the optimal expected value of the function $\mathsf{f}_{\mathsf{BWMP}}$. ◀

The following theorem shows that a polynomial time solution to our problem would lead to a polynomial time algorithm to solve mean-payoff games. The proof uses a reduction similar to the one used in the proof of Theorem 9.

▶ **Theorem 13.** *Given a two-player game G with an initial state s_{init}, we can construct in log-space an MDP Γ_G with an initial state s'_{init} such that $\mathbb{E}_{s'_{\mathit{init}}}^{\Gamma_G}(\mathsf{f}_{\mathsf{BWMP}}) = V_{s_{\mathit{init}}}^{\mathsf{f}_{\mathsf{Mean}}}(G)$.*

Finally, we show that in an MDP, the expected bounded window mean-payoff equals the supremum of the fixed window mean-payoff over all window lengths and over all strategies, which match the intuition behind these definitions.

▶ **Theorem 14.** *For every MDP Γ, we have* $\sup\limits_{\sigma \in \mathsf{strat}(\Gamma)} \mathbb{E}^{\Gamma[\sigma]}(\mathsf{f}_{\mathsf{BWMP}}) = \sup\limits_{\ell} \sup\limits_{\sigma \in \mathsf{strat}(\Gamma)} \mathbb{E}^{\Gamma[\sigma]}(\mathsf{f}_{\mathsf{WMP}}^{\ell})$.

5 Algorithms and Hardness for Direct Variants

We start with the direct fixed window objective. Surprisingly the complexity of solving this objective is substantially higher than its prefix-independent conterpart. Our algorithm is exponential in ℓ and in the number of bits to encode W. As shown later, the higher complexity is explained by the fact that the problem is PSpace-Hard.

▶ **Theorem 15.** *Given an MDP Γ with an initial state s_{init}, a window length ℓ and a threshold $\lambda \in \mathbb{Q}$, whether $\mathbb{E}^{\Gamma}_{s_{init}}(\mathsf{f}^{\ell}_{\mathsf{DirWMP}}) \geq \lambda$ can be decided in time $O(\mathsf{poly}(|S| \cdot W^{\ell} \cdot \ell^2))$ and deterministic exponential memory strategies suffice to play optimally.*

Proof sketch. As $\mathsf{f}^{\ell}_{\mathsf{DirWMP}}$ is prefix-dependent, it is not sufficient to know the expected value of this function in the MECs of Γ. Instead, we construct a new MDP Γ_ℓ which is a finite state structure that maps each infinite path π of Γ to the minimal mean-payoff encountered in a window of size ℓ along this path. The state space of Γ_ℓ is $S' = S \times ([W]_0)^{\ell-1} \times \Lambda$ where $\Lambda = \{\frac{p}{q} \mid q \in [\ell],\ p \in [q \cdot W]_0\}$ and the initial state $s'_{\mathsf{init}} = (s_{\mathsf{init}}, [W, \ldots, W], W)$. Informally, a state $t = (s, [w_1, \ldots, w_{\ell-1}], \lambda_t) \in S'$ summarizes all finite paths $\rho = s_0 \ldots s$ in Γ where the last $\ell - 1$ weights encountered are $w_1, \ldots, w_{\ell-1}$, and λ_t keeps track of the minimum window mean-payoff seen so far in π for window size ℓ. Moreover, in MDP Γ_ℓ every edge exiting t has a weight equal to λ_t. In this way, for each $\pi' \in \mathsf{Paths}^{\Gamma_\ell}$, the sequence of weights seen along π' is a non-increasing series of values belonging to the finite set Λ. Thus, eventually the sequence reaches a value λ which never changes again, this λ is the direct fixed window mean-payoff of the corresponding path in Γ and because every edge exiting t has a weight equal to λ_t, we see that λ is also the mean-payoff of π' in Γ_ℓ. Now, it remains to compute the optimal expected mean-payoff in Γ_ℓ which can be done in polynomial time in the size of Γ_ℓ using linear programming, see e.g.[16]. This optimal expected mean-payoff in Γ_ℓ is equal to the optimal expected direct fixed window mean-payoff for window size ℓ in Γ. Note that although the algorithm is exponential in ℓ and in the number of bits used to represent W, it is fixed parameter tractable, if we consider W and ℓ as parameters.

Since optimal expected mean-payoff in an MDP can be achieved using memoryless deterministic strategies and the size of Γ_ℓ is exponential in the size of the original MDP Γ, an optimal strategy with memory exponential in the size of Γ exists. ◀

We now provide the following hardness result:

▶ **Theorem 16.** *Given an MDP Γ with an initial state s_{init}, a window length ℓ and a $\lambda \in \mathbb{Q}$, deciding whether $\mathbb{E}^{\Gamma}_{s_{init}}(\mathsf{f}^{\ell}_{\mathsf{DirWMP}}) \geq \lambda$ is* PSPACE-HARD.

Proof. We show a reduction from the threshold probability problem for shortest path objectives [12]. An instance of the threshold probability problem is given by an MDP $\Gamma = (S, E, Act, s_{\mathsf{init}}, w, \mathbb{P})$ where w.l.o.g., we have that w assigns positive weights on the edges, $T \subseteq S$ is a set of target states, and for a strategy σ, the truncated sum $TS^T : Paths(\Gamma^{[\sigma]}) \longrightarrow \mathbb{N} \cup \infty$ up to T from the initial state s_{init} is defined as

$$TS^T(\rho) = \begin{cases} \sum_{i=0}^{n-1} w(e_i) & \text{if } \exists n \text{ such that } \rho(n) \in T \text{ and } \forall i \leq n-1, \text{ we have } \rho(i) \notin T \\ \infty & \text{if } \forall i \geq 0, \rho(i) \notin T, \end{cases}$$

where $e_i = (\rho(i), a, \rho(i+1))$, $a \in Act$; for a threshold $L \in \mathbb{N}$, and a probability threshold p, the problem asks to decide if there exists a strategy σ such that $\mathbb{P}_{\Gamma^{[\sigma]}, s_{\mathsf{init}}}[\{\rho \in Paths(\Gamma^{[\sigma]}) \mid TS^T(\rho) \leq L\}] \geq p$. The problem is known to be PSPACE-COMPLETE, even for acyclic MDPs [12]. The target set T is assumed to be made of absorbing states (i.e., with self-loops); the acyclicity is to be interpreted over the rest of the underlying graph.

Let $\Gamma = (S, E, Act, s_{\mathsf{init}}, w, \mathbb{P})$, where $S = T \uplus V$, and T is a set of target vertices. The acyclicity of that MDP implies that, from the initial state $s_{\mathsf{init}} \notin T$, it takes at most $|S| - 1$ steps to reach a vertex in T. Let W be the maximum weight appearing in Γ. We assume that $L \leq W \cdot (|S| - 1)$, otherwise the problem is trivial.

We construct a new MDP $\Gamma' = (S', E, Act', s_{\mathsf{init}}, w', \mathbb{P}')$ where $S' = S \cup \{s_{\mathsf{final}_1}, s_{\mathsf{final}_2}\}$, $Act' = Act \cup \{\mathsf{loop}, \alpha, \beta\}$. The set of edges $E' = \{(v, a, s) \mid (v, a, s) \in E,\ v \in V,\ s \in S\} \cup \{(t, \alpha, s_{\mathsf{final}_1}) \mid t \in T\} \cup \{(t, \beta, s_{\mathsf{final}_2}) \mid t \in T\} \cup \{(s_{\mathsf{final}_1}, \mathsf{loop}, s_{\mathsf{final}_1})\} \cup \{(s_{\mathsf{final}_2}, \mathsf{loop}, s_{\mathsf{final}_2})\}$ $\cup \{(v, \beta, s_{\mathsf{final}_2}) \mid v \in V$ and there is no outgoing edge from v in $\Gamma\}$. The probability function $\mathbb{P}'$ is defined as:

- $\mathbb{P}'(v, a, s) = \mathbb{P}(v, a, s)$ such that $(v, a, s) \in E,\ v \in V,\ s \in S$;
- $\mathbb{P}'(t, \alpha, s_{\mathsf{final}_1}) = 1$ for $t \in T$;
- $\mathbb{P}'(t, \beta, s_{\mathsf{final}_2}) = 1$ for $t \in T$;
- $\mathbb{P}'(s_{\mathsf{final}_j}, \mathsf{loop}, s_{\mathsf{final}_j}) = 1$ for $j \in \{1, 2\}$;
- $\mathbb{P}'(v, \beta, s_{\mathsf{final}_2}) = 1$ for $(v, \beta, s_{\mathsf{final}_2}) \in E'$ and $v \in V$;

The weight function w' is defined as follows.

- $w'(v, a, s) = -w(v, a, s)$ such that $(v, a, s) \in E,\ v \in V,\ s \in S$;
- $w'(t, \alpha, s_{\mathsf{final}_1}) = L$, for $t \in T$;
- $w'(t, \beta, s_{\mathsf{final}_2}) = W \cdot (|S| - 1)$, for $t \in T$;
- $w'(s_{\mathsf{final}_1}, \mathsf{loop}, s_{\mathsf{final}_1}) = 0$;
- $w'(s_{\mathsf{final}_2}, \mathsf{loop}, s_{\mathsf{final}_2}) = -\frac{1}{|S|}$, and
- $w'(v, \beta, s_{\mathsf{final}_2}) = W \cdot (|S| - 1)$ for $(v, \beta, s_{\mathsf{final}_2}) \in E'$ and $v \in V$.

Let $\ell = |S|$. Starting from s_{init}, since the weights on all the edges on the paths leading to a state in $t \in T$ are negative, the direct fixed window mean-payoff will consider paths until they reach s_{final_j} for $j \in \{1, 2\}$ given that the weights on the edges outgoing from t are positive.

We now call a path to be *good* if t appears in the path for some $t \in T$, and the sum of the edges from s_{init} to t is at least $-L$, otherwise the path is *bad.* Note that for a good path, choosing α leads to a direct fixed window mean-payoff of 0, while choosing β leads to direct fixed window mean-payoff of $-\frac{1}{|S|}$. On the other hand, for a bad path, choosing α gives a direct fixed window mean-payoff of at most $-\frac{1}{|S|}$, while choosing β gives a direct fixed window mean-payoff of $-\frac{1}{|S|}$. Therefore, for an optimal strategy, the direct fixed window mean-payoff for a *good* path is 0, and for a *bad* path, it is $-\frac{1}{|S|}$.

We have $|\Gamma'| = O(\mathsf{poly}(|\Gamma|))$. Furthermore, the expected value of the direct fixed window mean-payoff, $\mathbb{E}^{\Gamma}_{s_{\mathsf{init}}}(\mathsf{f}^{\ell}_{\mathsf{DirWMP}}) \geq p \cdot 0 + (1-p) \cdot -\frac{1}{|S|} = -(1-p) \cdot \frac{1}{|S|}$ iff there is a solution to the threshold probability problem.

Note that since $\ell = |S|$, deciding whether the expected value of the direct fixed window mean-payoff for an MDP is greater than or equal to some threshold is PSPACE-HARD even when ℓ is given in unary. Thus, we cannot expect to have an algorithm that is polynomial in the value of ℓ unless P=PSPACE[5]. ◀

We now consider the bounded case. In fact, the function $\mathsf{f}_{\mathsf{DirBWMP}}$ is equivalent to $\mathsf{f}_{\mathsf{BWMP}}$:

▶ **Lemma 17.** *For every path π in an MDP, we have that* $\mathsf{f}_{\mathsf{DirBWMP}}(\pi) = \mathsf{f}_{\mathsf{BWMP}}(\pi)$.

Proof sketch. It is easy to see that $\mathsf{f}_{\mathsf{DirBWMP}}(\pi) \leq \mathsf{f}_{\mathsf{BWMP}}(\pi)$. Now for every $\varepsilon > 0$, a window mean-payoff value of $\mathsf{f}_{\mathsf{BWMP}}(\pi) - \varepsilon$ can be ensured from the beginning of the path π by considering appropriately large window length. Since $\mathsf{f}_{\mathsf{DirBWMP}}(\pi)$ is the supremum of the window mean-payoff values that can be ensured with arbitrarily large window lengths, the result follows. ◀

[5] The reduction does not work for Markov chains since we cannot get a threshold for the window mean-payoff that separates the cases when there is a solution to the threshold probability problem for shortest path objective and when a solution to the problem does not exist. That is, if the sum of path from s'_{init} to t is below L and the edge corresponding to action α is taken in t, we do not know how much below 0 will the window mean-payoff be.

As a direct corollary of Lemma 17, Theorem 10 and Theorem 13, we obtain:

▶ **Theorem 18.** *Given an MDP Γ and a $\lambda \in \mathbb{Q}$, we have $\mathbb{E}^{\Gamma}_{s_{init}}(\mathsf{f}_{\mathsf{DirBWMP}}) = \mathbb{E}^{\Gamma}_{s_{init}}(\mathsf{f}_{\mathsf{BWMP}})$, and whether $\mathbb{E}^{\Gamma}_{s_{init}}(\mathsf{f}_{\mathsf{DirBWMP}}) \geq \lambda$ can be decided in* UP $\cap$ coUP*, and it is as hard as solving two-player mean-payoff games.*

6 Solving Window Mean-Payoff Objectives for Markov Chain

We focus on the bounded window objective and the direct fixed window objective for MCs, as MCs are special cases of MDPs, and for these two objectives, we show strict improvement in the complexity of the algorithms compared to MDPs. We start with the bounded window mean-payoff function, for which we provide a polynomial time solution while the case of MDPs is at least as hard as mean-payoff games (Theorem 13).

▶ **Theorem 19.** *Given an MC $\mathcal{M}$ and a threshold $\lambda \in \mathbb{Q}$, whether $\mathbb{E}^{\mathcal{M}}_{s_{init}}(\mathsf{f}_{\mathsf{BWMP}}) \geq \lambda$ can be decided in polynomial time.*

We first outline the case of a BSCC $\mathcal{B}$ since by Proposition 1, each path in an MC almost surely ends up in a BSCC. Let $\lambda_{\mathcal{B}}$ be the expected value of $\mathsf{f}_{\mathsf{BWMP}}$ in $\mathcal{B}$:

▶ **Lemma 20.** *For an MC that is a BSCC $\mathcal{B}$, we have $\lambda_{\mathcal{B}} = \min_{\rho \in \mathsf{ElemCycles}(\mathcal{B})} \mathsf{MP}(\rho)$.*

Proof sketch. For every ℓ, a path π in $\mathcal{B}$ will almost surely have infinitely many infixes of length ℓ going around a minimum mean-cycle, leading to $\mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi) \leq c_{\mathcal{B}}$ where $c_{\mathcal{B}} = \min_{\rho \in \mathsf{ElemCycles}(\mathcal{B})} \mathsf{MP}(\rho)$. Moreover, for each path π in $\mathcal{B}$ and for every $\varepsilon > 0$, by choosing an appropriate window length ℓ, we have $\mathsf{f}^{\ell}_{\mathsf{WMP}}(\pi) \geq c_{\mathcal{B}} - \varepsilon$. By definition of $\mathsf{f}_{\mathsf{BWMP}}$, we have $\mathsf{f}_{\mathsf{BWMP}}(\pi) = c_{\mathcal{B}}$ almost surely. ◀

Proof sketch of Theorem 19. Note that $\mathbb{E}^{\mathcal{M}}_{s_{init}}(\mathsf{f}_{\mathsf{BWMP}}) = \sum_{\mathcal{B} \in \mathsf{BSCC}(\mathcal{M})} \mathsf{Pr}(\Diamond \mathcal{B}) \cdot \lambda_{\mathcal{B}}$. Since for each BSCC $\mathcal{B}$, both mean of the minimum mean-cycle in $\mathcal{B}$ and the probability of reaching $\mathcal{B}$ can be computed in polynomial time [15, 16], we obtain the result. ◀

We now consider the direct fixed window mean-payoff function. We show the following.

▶ **Theorem 21.** *Given an MC $\mathcal{M}$, with set S of states, a window length ℓ and a threshold $\lambda \in \mathbb{Q}$, whether $\mathbb{E}^{\mathcal{M}}_{s_{init}}(\mathsf{f}^{\ell}_{\mathsf{DirWMP}}) \geq \lambda$ can be decided in $O(\mathsf{poly}(|S| \cdot \ell \cdot W))$ time.*

We first consider the inductive property of windows (see [8]). For an infinite path $\pi = s_0 \ldots$, a threshold $\lambda \in \mathbb{Q}$, a window length ℓ, a position $i \in \mathbb{N}$ and $l \in [\ell]$, we say that the window starting at position i is *closed* at position $i+l$ with respect to λ if $\mathsf{WMP}^{\ell}(\pi(i, \infty)) \geq \lambda$. Otherwise, the window is *open*.

Inductive property of windows. Let $\pi = s_0 \ldots \in \mathsf{Paths}^{\mathcal{M}}$, ℓ be a window length, and λ be a threshold. Assume that a window starting at a position j is open at $j' < j + \ell$ but closed at $j' + 1$. Then, any window starting at a position between j and j' is closed at $j' + 1$.

Note that we cannot focus only on the BSCCs here. Let $\mathsf{f} = \mathsf{f}^{\ell}_{\mathsf{DirWMP}}$. Then, for every path $\pi \in \mathsf{Paths}^{\mathcal{M}}$, we have $\mathsf{f}(\pi) \in \Lambda$ with $\Lambda = \{\frac{p}{q} \mid q \in [\ell],\ p \in [q \cdot W]_0\}$. Let $\Lambda = \{\lambda_0, \ldots, \lambda_n\}$. For every $\lambda_i \in \Lambda$, we construct a new Markov chain $\mathcal{M}^{\lambda_i}_{\ell}$ so that the probability $\mathsf{Pr}(\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, [\lambda_i, \infty[))$ is equal to the probability of not reaching a trap state in $\mathcal{M}^{\lambda_i}_{\ell}$. Thanks to the inductive property of windows, we only need to remember the location of the largest window that is still open, as well as the "amount of payoff" that is required to close it. Hence, in the Markov chain $\mathcal{M}_\ell^{\lambda_i}$, the state space $S' = (S \times [\ell-1]_0 \times [W \cdot (\ell-1)]_0) \cup \{\mathsf{trap}\}$. If the window cannot be closed within ℓ steps, then the state trap is reached. For $\lambda_i \in \Lambda$, we have the following lemma.

▶ **Lemma 22.** $\mathsf{Pr}(\mathsf{f}^{-1}(\mathcal{M}, s_{init}, [\lambda_i, \infty[)) = \mathsf{Pr}(\pi \in \mathsf{Paths}^{\mathcal{M}_\ell^{\lambda_i}} \mid \pi \models \neg\Diamond\{\mathsf{trap}\})$

We can now prove Theorem 21.

Proof sketch of Theorem 21. Assume w.l.o.g. that, in Λ, we have $\lambda_0 < \ldots < \lambda_n$. Now for all $i \leq n-1$, we have $\mathsf{Pr}(\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, \lambda_i)) = \mathsf{Pr}(\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, [\lambda_i, \infty[)) - \mathsf{Pr}(\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, [\lambda_{i+1}, \infty[))$, and $\mathbb{E}^{\mathcal{M}}_{s_{\mathsf{init}}}(\mathsf{f}) = \sum\limits_{i=0}^{n} \mathsf{Pr}(\mathsf{f}^{-1}(\mathcal{M}, s_{\mathsf{init}}, \lambda_i)) \cdot \lambda_i$. Note that $|\Lambda| \leq \ell \cdot W \cdot \ell$ and for each $\lambda_i \in \Lambda$, we have that $|\mathcal{M}_\ell^{\lambda_i}| \leq |\mathcal{M}| \cdot \ell \cdot W \cdot \ell + 1$. Since reachability in Markov chain (here to the trap state) can be decided in polynomial time and W is given in binary, the result follows. ◀

If W is a parameter, we get a fixed parameter tractable algorithm.

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

2 Benjamin Bordais, Shibashis Guha, and Jean-François Raskin. Expected Window Mean-Payoff. *CoRR*, abs/1812.09298, 2018. arXiv:1812.09298.

3 Tomás Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. *Logical Methods in Computer Science*, 10(1), 2014.

4 Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Trading performance for stability in Markov decision processes. *J. Comput. Syst. Sci.*, 84:144–170, 2017.

5 Tomás Brázdil, Vojtech Forejt, Antonín Kucera, and Petr Novotný. Stability in Graphs and Games. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 10:1–10:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

6 Thomas Brihaye, Florent Delgrange, Youssouf Oualhadj, and Mickael Randour. Life is Random, Time is Not: Markov Decision Processes with Window Objectives. *CoRR*, abs/1901.03571, 2019. arXiv:1901.03571.

7 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. On the Complexity of Heterogeneous Multidimensional Games. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

8 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at Mean-Payoff and Total-Payoff through Windows. In *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2013.

9 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015.

10 Krishnendu Chatterjee and Monika Henzinger. Efficient and Dynamic Algorithms for Alternating Büchi Games and Maximal End-Component Decomposition. *J. ACM*, 61(3):15:1–15:40, June 2014.

11 Luca De Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, Stanford, CA, USA, 1998.

12 Christoph Haase and Stefan Kiefer. The Odds of Staying on Budget. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 234–246, 2015.

13 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Looking at mean payoff through foggy windows. *Acta Inf.*, 55(8):627–647, 2018.

14 Marcin Jurdzinski. Deciding the Winner in Parity Games is in UP $\cap$ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.

15 Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

16 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

17 Moshe Y. Vardi. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 327–338. IEEE Computer Society, 1985.

18 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.

Interval Temporal Logic for Visibly Pushdown Systems

Laura Bozzelli
University of Napoli “Federico II”, Napoli, Italy

Angelo Montanari
University of Udine, Udine, Italy

Adriano Peron
University of Napoli “Federico II”, Napoli, Italy

Abstract

In this paper, we introduce and investigate an extension of Halpern and Shoham's interval temporal logic HS for the specification and verification of branching-time context-free requirements of pushdown systems under a state-based semantics over Kripke structures. Both homogeneity and visibility are assumed. The proposed logic, called nested BHS, supports branching-time both in the past and in the future, and is able to express non-regular properties of linear and branching behaviours of procedural contexts in a natural way. It strictly subsumes well-known linear time context-free extensions of LTL such as CaRet [4] and NWTL [2]. The main result is the decidability of the visibly pushdown model-checking problem against nested BHS. The proof exploits a non-trivial automata-theoretic construction.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Pushdown systems, Interval temporal logic, Model checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.33

1 Introduction

Model checking in the framework of interval temporal logics. *Point-based* temporal logics (PTLs), such as, for instance, the linear-time temporal logic LTL [25] and the branching-time temporal logics CTL and CTL* [16], provide a standard framework for the specification and verification (model checking) of the behavior of reactive systems. In this framework, the evolution of a system over time is described state-by-state (“point-wise” view). *Interval temporal logics* (ITLs) have been proposed as an alternative setting for reasoning about time [17, 24, 28]. Unlike standard PTLs, they assume intervals, instead of points, as their primitive entities. ITLs allow one to specify relevant temporal properties that involve, e.g., actions with duration, accomplishments, and temporal aggregations, which are inherently “interval-based”, and thus cannot be naturally expressed by PTLs. They have been applied in various areas of computer science, including formal verification, computational linguistics, planning, and multi-agent systems (e.g. see [18, 24, 26]). Among ITLs, the landmark is *Halpern and Shoham's modal logic of time intervals* HS [17], which features one modality for each of the 13 ordering relations between pairs of intervals (the so-called Allen's relations), apart from equality. The satisfiability problem for HS and most of its fragments is undecidable over all relevant classes of linear orders, with some meaningful exceptions (see [23, 12, 13]).

In the last years, the model checking problem for HS over *finite* Kripke structures (*finite MC problem*) has been extensively studied [8, 9, 10, 18, 19, 20, 21, 22]. Each *finite* path of a Kripke structure is interpreted as an interval, whose labelling is defined on the basis of the labelling of the component states. In particular, the finite MC problem under the *homogeneity assumption* (a proposition letter holds over an interval if and only if it holds over each component state) and the *state-based semantics* (time branches both in the future and

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 33; pp. 33:1–33:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the past) has been investigated in [9, 21, 22]. In this setting, it turns out to be decidable and the complexity of the problem for full HS and *its syntactical fragments* has been almost completely settled (the only intriguing open question is the complexity of the problem for full HS currently located in between an Expspace lower bound and a non-elementary upper bound). In [10] the authors study the expressiveness of the state-based semantics of HS and of two variants: the *computation-tree-based semantics*, that allows time to branch only in the future, and the *trace-based semantics*, that disallows time branching. The computation-tree-based variant of HS is expressively equivalent to finitary CTL^* (the variant of CTL^* with quantification over finite paths), while the trace-based variant is equivalent to LTL (but at least exponentially more succinct). The state-based variant is more expressive than the computation-tree-based variant and expressively incomparable with both LTL and CTL^*.

Model checking of pushdown systems. In the last two decades, model checking of pushdown automata (PDA) against non-regular properties has received a lot of attention [2, 4, 5, 11, 14]. PDA are an infinite-state formalism suitable to model the control flow of typical sequential programs with recursive procedure calls. PDA have a decidable model-checking problem against regular specifications (e.g. see [29, 15]) but the general problem of checking context-free properties is undecidable. The latter problem has been positively solved, however, for interesting subclasses of context-free requirements such as those expressed by the linear temporal logic CaRet [4], a context-free extension of LTL. CaRet formulas are interpreted on words over a *pushdown alphabet* which is partitioned into three disjoint sets of *call*, *return*, and *internal* symbols respectively denoting a procedure invocation (i.e., a push stack operation), a return from a procedure call (a pop stack operation), and an internal operation (not affecting the stack). CaRet allows one to specify non-regular context-free properties which require matching of calls and returns such as correctness of procedures with respect to pre and post conditions, and security properties that require the inspection of the call-stack.
An automata-theoretic generalization of CaRet is the class of *Nondeterministic Visibly Pushdown Automata* (NVPA) [5], a subclass of PDA where the operations on the stack are determined by the input symbols over a pushdown alphabet. The accepted class of *visibly pushdown languages* (or VPL) is closed under Boolean operations, and the problem of language inclusion, which is undecidable for context-free languages, is instead decidable for VPL. This implies that under the *visibility requirement* (call and returns are made visible) the model-checking problem of pushdown systems (*VPMC* problem) against *linear-time* pushdown properties is decidable. To the best of our knowledge, the only *branching-time* context-free logic introduced in the literature with a decidable *VPMC* problem (in particular, the problem is Exptime-complete) is the *visibly pushdown μ-calculus* (VP-μ) [3], an extension of the modal μ-calculus with future modalities which allow one to specify requirements on the branching behavior of procedural contexts.

Paper contributions. First of all, we unify the linear time (trace-based) and branching time (computation-based and state-based) semantic variants of HS in a common framework. To this end, we extend HS with a novel *binding operator*, which restricts the evaluation of a formula to the interval sub-structure induced by the current interval. The extension is denoted by BHS. By additionally generalizing the interval mapping also to infinite paths, the logic BHS gives one the ability to force a linear-time semantics of the temporal modalities along a (finite or infinite) path, so that the trace-based semantics can be subsumed.

As a second contribution, BHS, with the state-based semantics, is further extended for specifying branching-time context-free requirements of pushdown systems under the homogeneity and visibility assumptions. It is the very first time (as far as we know) that

HS is studied in the context of model checking of pushdown systems (in general, of infinite state systems) where only PTL approaches have been investigated, and it is interesting to notice that the most distinctive feature of pushdown systems, namely, the matching of a call with the corresponding return, has a natural interval nature (it bounds meaningful computation intervals where local properties can be checked). This suggests that an *interval* temporal logic (instead of point-based one) could be a natural choice. The extension of BHS we propose, called *nested* BHS, is powerful despite its simplicity: we just add to BHS a special proposition p_{wm} that captures finite intervals corresponding to computations with well-matched pairs of calls and returns. We investigate the expressiveness of nested BHS showing that it strictly subsumes well-known linear time context-free extensions of LTL such as CaRet [4] and NWTL [2]. Nested BHS is a formalism which supports past and future branching-time besides linear time: future branching time allows to express context-free versions of standard CTL*-like properties (for instance, multiple return conditions for procedure calls); past branching time allows to check properties (regular and context-free) of multiple histories leading to a common fixed state. An expressiveness comparison between VP-μ and BHS is out of the scopes of this paper. Here, we just observe that while VP-μ is bisimulation-closed [3], HS, and thus BHS, with the state-based semantics, is not (this is due to branching past [10]). Whether the future fragment of BHS is subsumed or not by VP-μ is an intriguing open issue.

As a third and main result, we prove that the *VPMC* problem against nested BHS is decidable, although with a non-elementary complexity. For the upper bound, we exploit a non-trivial automata-theoretic approach consisting in translating a nested BHS formula ψ into an NVPA accepting suitable encodings of the computations of the given pushdown system which satisfy formula ψ. Actually, we conjecture that the non-elementary complexity of nested BHS only depends on the nesting depth of the binding modality.

2 Interval temporal logic HS with binding contexts

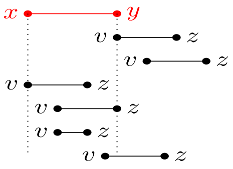
In this section, we introduce the temporal logic HS with binding contexts (BHS for short) and the model-checking framework for verifying BHS formulas.

We fix the following notation. Let $\mathbb{N}$ be the set of natural numbers. For all $i, j \in \mathbb{N}$, with $i \leq j$, $[i, j]$ denotes the set of natural numbers h such that $i \leq h \leq j$. Let w be a finite or infinite word over some alphabet. We denote by $|w|$ the length of w (we set $|w| = \infty$ if w is infinite). For all $i, j \in \mathbb{N}$, with $i \leq j$, $w(i)$ is the i-th letter of w, $w^i = w(i)w(i+1)\ldots$ is the suffix of w from position i on, while $w[i, j]$ denotes the infix of w given by $w(i)\cdots w(j)$. The set $\text{Pref}(w)$ of *proper prefixes* of w is the set of non-empty finite words u such that $w = u \cdot v$ for some non-empty word v. The set $\text{Suff}(w)$ of *proper suffixes* of w is the set of non-empty words u such that $w = v \cdot u$ for some non-empty finite word v. We fix a finite set $\mathcal{AP}$ of atomic propositions which represent predicates over the states of the given system.

A *Kripke structure* over $\mathcal{AP}$ is a tuple $\mathcal{K} = (\mathcal{AP}, S, E, \mathit{Lab}, s_0)$, where S is a set of states, $E \subseteq S \times S$ is a transition relation, $\mathit{Lab} : S \mapsto 2^{\mathcal{AP}}$ is a labelling function assigning to each state s the set of propositions that hold over it, and $s_0 \in S$ is the initial state. We say that $\mathcal{K}$ is finite if S is finite. A path π of $\mathcal{K}$ is a non-empty word over S such that, for all $0 \leq i < |\pi|$, $(\pi(i), \pi(i+1)) \in E$. A path is *initial* if it starts from the initial state of $\mathcal{K}$. A path π induces the word $\mathit{Lab}(\pi)$ over $2^{\mathcal{AP}}$ having the same length as $|\pi|$ given by $\mathit{Lab}(\pi(0))\mathit{Lab}(\pi(1))\ldots$. We also say that $\mathit{Lab}(\pi)$ is the trace induced by π.

An interval algebra to reason about intervals and their relative orders was proposed by Allen in [1], while a systematic logical study of interval representation and reasoning was done a few years later by Halpern and Shoham, who introduced the interval temporal logic

Table 1 Allen's relations and corresponding HS modalities.

Allen relation	HS	Definition w.r.t. interval structures	Example
MEETS	$\langle A\rangle$	$[x,y]\mathcal{R}_A[v,z] \iff y=v$	
BEFORE	$\langle L\rangle$	$[x,y]\mathcal{R}_L[v,z] \iff y<v$	
STARTED-BY	$\langle B\rangle$	$[x,y]\mathcal{R}_B[v,z] \iff x=v \wedge z<y$	
FINISHED-BY	$\langle E\rangle$	$[x,y]\mathcal{R}_E[v,z] \iff y=z \wedge x<v$	
CONTAINS	$\langle D\rangle$	$[x,y]\mathcal{R}_D[v,z] \iff x<v \wedge z<y$	
OVERLAPS	$\langle O\rangle$	$[x,y]\mathcal{R}_O[v,z] \iff x<v<y<z$	

HS featuring one modality for each Allen relation, but equality [17]. Table 1 depicts 6 of the 13 Allen's relations, together with the corresponding HS (existential) modalities. The other 7 relations are the 6 inverse relations (given a binary relation $\mathcal{R}$, the inverse relation $\overline{\mathcal{R}}$ is such that $b\overline{\mathcal{R}}a$ iff $a\mathcal{R}b$) and equality. Here, we introduce an extension of the logic HS, called *binding* HS (BHS for short), obtained by adding a novel *binding modality* which allows one to restrict the valuation of a formula to the interval sub-model induced by a given interval.

Let $\mathcal{AP}_u$ be a finite set of uninterpreted interval properties. BHS formulas ψ over $\mathcal{AP}_u$ are defined by the grammar:

$$\psi ::= \texttt{true} \mid \texttt{false} \mid p_u \mid \neg\psi \mid \psi \wedge \psi \mid \langle X\rangle\psi \mid \mathbb{B}\psi$$

where $p_u \in \mathcal{AP}_u$, $\langle X\rangle$ is the existential temporal modality for the (non-trivial) Allen's relation $X \in \{A, L, B, E, D, O, \overline{A}, \overline{L}, \overline{B}, \overline{E}, \overline{D}, \overline{O}\}$, and $\mathbb{B}$ is the unary *binding modality*. The size $|\psi|$ of a formula ψ is the number of distinct subformulas of ψ. We also exploit the standard logical connectives $\vee$ (disjunction) and $\rightarrow$ (implication) as abbreviations, and for any temporal modality $\langle X\rangle$, the dual universal modality $[X]$ defined as: $[X]\psi := \neg\langle X\rangle\neg\psi$. The standard logic HS is obtained from BHS by disallowing the binding modality.

W.l.o.g. we assume the *non-strict semantics*, which admits intervals consisting of a single point. Under such an assumption, all HS-temporal modalities can be expressed in terms of $\langle B\rangle, \langle E\rangle, \langle\overline{B}\rangle$, and $\langle\overline{E}\rangle$ [28]. As an example, $\langle A\rangle$ can be expressed in terms of $\langle E\rangle$ and $\langle\overline{B}\rangle$ as: $\langle A\rangle\,\varphi := (\neg\,\langle E\rangle\,\texttt{true} \wedge (\varphi \vee \langle\overline{B}\rangle\,\varphi)) \vee \langle E\rangle(\neg\,\langle E\rangle\,\texttt{true} \wedge (\varphi \vee \langle\overline{B}\rangle\,\varphi))$. BHS can be viewed as an extension, by means of the binding modality $\mathbb{B}$, of a multi-modal logic where $\langle B\rangle, \langle E\rangle, \langle\overline{B}\rangle$, and $\langle\overline{E}\rangle$ are the primitive temporal modalities. BHS formulas can thus be interpreted over a multi-modal Kripke structure, called *abstract interval model* (AIM for short), where intervals are treated as atomic objects and Allen's relations as binary relations over intervals. As we will see, in model-checking against BHS, a Kripke structure is suitably mapped to an AIM.

Formally, an *abstract interval model* (AIM) [21] over $\mathcal{AP}_u$ is a tuple $\mathcal{A} = (\mathcal{AP}_u, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, Lab_{\mathbb{I}})$, where $\mathbb{I}$ is a possibly infinite set of worlds (abstract intervals), $B_{\mathbb{I}}$ and $E_{\mathbb{I}}$ are two binary relations over $\mathbb{I}$, and $Lab_{\mathbb{I}} : \mathbb{I} \mapsto 2^{\mathcal{AP}_u}$ is a labeling function, which assigns a set of proposition letters from $\mathcal{AP}_u$ to each abstract interval. In the interval setting, $\mathbb{I}$ is interpreted as a set of intervals and $B_{\mathbb{I}}$ and $E_{\mathbb{I}}$ as Allen's relations B (*started-by*) and E (*finished-by*), respectively; $Lab_{\mathbb{I}}$ assigns to each interval in $\mathbb{I}$ the set of atomic propositions that hold over it. The semantics of the $\mathbb{B}$ modality is based on the notion of *abstract interval sub-model* induced by a given abstract interval.

▶ **Definition 1.** *Let $\mathcal{A} = (\mathcal{AP}_u, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, Lab_{\mathbb{I}})$ be an AIM. The* sub-interval relation *$G_{\mathbb{I}}$ induced by $B_{\mathbb{I}}$ and $E_{\mathbb{I}}$ is defined as follows: $(I, J) \in G_{\mathbb{I}}$ iff $(I, J) \in (B_{\mathbb{I}} \cup E_{\mathbb{I}})^*$ (i.e., (I, J) is in the reflexive and transitive closure of the relation $B_{\mathbb{I}} \cup E_{\mathbb{I}}$). For $I \in \mathbb{I}$, the* abstract interval sub-model induced by *I is the AIM $\mathcal{A}^I = (\mathcal{AP}_u, \mathbb{I}^I, B^I_{\mathbb{I}}, E^I_{\mathbb{I}}, Lab^I_{\mathbb{I}})$, where $\mathbb{I}^I$ is the set of abstract sub-intervals of I, i.e., the set of $J \in \mathbb{I}$ such that $(I, J) \in G_{\mathbb{I}}$ and $B^I_{\mathbb{I}}$ (resp., $E^I_{\mathbb{I}}$, $Lab^I_{\mathbb{I}}$) is the restriction of $B_{\mathbb{I}}$ (resp., $E_{\mathbb{I}}$, $Lab_{\mathbb{I}}$) to $\mathbb{I}^I$.*

Semantics of BHS. Let $\mathcal{A} = (\mathcal{AP}_u, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, Lab_{\mathbb{I}})$ be an AIM. A context C is either ε (the empty context) or an abstract interval $J \in \mathbb{I}$. We write $\mathcal{A}^{\varepsilon}$ for $\mathcal{A}$ (the meaning of $\mathbb{I}^{\varepsilon}$, $B_{\mathbb{I}}^{\varepsilon}$, $E_{\mathbb{I}}^{\varepsilon}$, and $Lab_{\mathbb{I}}^{\varepsilon}$ is analogous). For an interval $I \in \mathbb{I}^C$ and a BHS formula ψ, the satisfaction relation $\mathcal{A}^C, I \models \psi$ is inductively defined as follows (the Boolean connectives are treated as usual):

- $\mathcal{A}^C, I \models p_u$ iff $p_u \in Lab_{\mathbb{I}}^C(I)$, for any $p_u \in \mathcal{AP}_u$;
- $\mathcal{A}^C, I \models \langle X \rangle \psi$, for $X \in \{B, E\}$, iff $I\, X_{\mathbb{I}}^C\, J$ and $\mathcal{A}^C, J \models \psi$ for some $J \in \mathbb{I}^C$;
- $\mathcal{A}^C, I \models \langle \overline{X} \rangle \psi$, for $\overline{X} \in \{\overline{B}, \overline{E}\}$, iff $J\, X_{\mathbb{I}}^C\, I$ and $\mathcal{A}^C, J \models \psi$ for some $J \in \mathbb{I}^C$;
- $\mathcal{A}^C, I \models \mathbb{B}\psi$ iff $\mathcal{A}^I, I \models \psi$.

Following [21], we propose a state-based approach for model-checking Kripke structures against BHS which consists in defining a mapping from Kripke structures to AIMs, where the abstract intervals correspond to the paths of the Kripke structure and the following two assumptions are adopted: (i) the set $\mathcal{AP}_u$ of HS-propositions coincides with the set $\mathcal{AP}$ of proposition letters for the given Kripke structure, and (ii) a proposition holds over an interval if and only if it holds over all its subintervals (*homogeneity principle*). Differently from [21], where only finite paths are considered, here we consider both finite and infinite paths.

▶ **Definition 2.** *Let $\mathcal{K} = (\mathcal{AP}, S, E, Lab, s_0)$ be a Kripke structure. The AIM induced by $\mathcal{K}$ is $\mathcal{A}_{\mathcal{K}} = (\mathcal{AP}, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, Lab_{\mathbb{I}})$, where $\mathbb{I}$ is the set of finite and infinite paths of $\mathcal{K}$, and:*

- $B_{\mathbb{I}} = \{(\pi, \pi') \in \mathbb{I} \times \mathbb{I} \mid \pi' \in \mathrm{Pref}(\pi)\}$, $E_{\mathbb{I}} = \{(\pi, \pi') \in \mathbb{I} \times \mathbb{I} \mid \pi' \in \mathrm{Suff}(\pi)\}$, *and*
- *for all* $p \in \mathcal{AP}$, $Lab_{\mathbb{I}}^{-1}(p) = \{\pi \in \mathbb{I} \mid p \in \bigcap_{i=0}^{i<|\pi|} Lab(\pi(i))\}$.

A Kripke structure $\mathcal{K}$ over $\mathcal{AP}$ is a *model* of a BHS formula ψ over $\mathcal{AP}$, written $\mathcal{K} \models \psi$, if for all *initial* paths π of $\mathcal{K}$, $\mathcal{A}_{\mathcal{K}}, \pi \models \psi$. The *finite* model-checking problem consists in checking whether $\mathcal{K} \models \psi$, for a given BHS formula ψ and a finite Kripke structure $\mathcal{K}$.

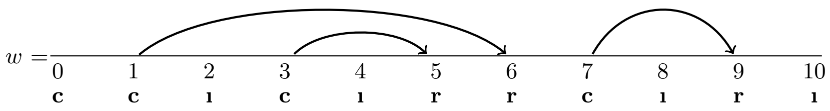
We observe that in the considered model-checking setting, the semantics of temporal modalities $\langle \mathrm{B} \rangle$ and $\langle \mathrm{E} \rangle$ is "linear-time" both in HS and in BHS, i.e., $\langle \mathrm{B} \rangle$ and $\langle \mathrm{E} \rangle$ allow one to select only subpaths (proper prefixes and suffixes) of the current timeline (computation). As for the temporal modalities $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$, while in HS the semantics of these modalities is always "branching-time" (i.e., $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$ allow one to non-deterministically extend the current timeline in the future and in the past, respectively), in BHS the semantics of $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$ can be either "linear-time" or "branching-time", depending on the current context.

Forcing linear time. We now show how the binding modality can be used to force a linear time semantics for a formula. By exploiting the notion of abstract interval sub-model, the linear-time model-checking setting for HS formulas introduced in [10] can be reformulated as follows: $\mathcal{K}$ is a model of an HS formula ψ under the linear-time (or *trace-based*) semantics, written $\mathcal{K} \models_{\mathsf{lin}} \psi$, if for all initial infinite paths π of $\mathcal{K}$ and positions $i \geq 0$, $\mathcal{A}_{\mathcal{K}}^{\pi}, \pi[0, i] \models \psi$. It is easy to check that $\mathcal{K} \models_{\mathsf{lin}} \psi$ iff $\mathcal{K} \models \mathbb{B}((\neg \langle \mathrm{A} \rangle \texttt{true}) \to [B]\psi)$ where the subformula $\neg \langle \mathrm{A} \rangle \texttt{true}$ captures the infinite paths π and the binding modality $\mathbb{B}$ forces the occurrences of $\langle \overline{\mathrm{B}} \rangle$ and $\langle \overline{\mathrm{E}} \rangle$ in ψ to refer only to sub-paths of π.

3 Model Checking Visibly Pushdown Systems against nested BHS

In this section we introduce and address expressiveness issues of a context-free extension of BHS, called *nested BHS*, for model checking (infinite-state) Kripke structures generated by *Visibly Pushdown Systems* (VPS).

We first recall the standard notions of *pushdown alphabet* and VPS. A *pushdown alphabet* is a finite alphabet $\Sigma = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$ which is partitioned into a set Σ_{call} of *calls*, a set Σ_{ret} of *returns*, and a set Σ_{int} of *internal actions*. This partition induces a nested hierarchical structure in a word over Σ obtained by associating to each call the corresponding matching return (if any) in a well-nested manner. Formally, the set of *well-matched finite words* w_m over Σ is inductively defined by the following abstract syntax: $w_m := \varepsilon \mid a \cdot w_m \mid c \cdot w_m \cdot r \cdot w_m$, where ε is the empty word, $a \in \Sigma_{int}$, $c \in \Sigma_{call}$, and $r \in \Sigma_{ret}$. Let w be a non-empty word over Σ. For a call position $0 \leq i < |w|$, if there is $j > i$ such that j is a return position of w and $w[i+1, j-1]$ is a well-matched finite 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. An *infinite word is well-matched* if each call (resp., return) has a matching return (resp., matching call). For instance, consider the finite word w depicted below where $\Sigma_{call} = \{c\}$, $\Sigma_{ret} = \{r\}$, and $\Sigma_{int} = \{ı\}$. Note that 0 is the unique unmatched call position of w.



To verify recursive programs, we assume that the set $\mathcal{AP}$ of atomic propositions (which represent predicates over the states of the system) contains three special propositions, namely, *call*, *ret*, and *int*: *call* denotes the invocation of a procedure, *ret* denotes the return from a procedure, and *int* denotes internal actions of the current procedure. Under this assumption, the set $\mathcal{AP}$ induces a pushdown alphabet $\Sigma_{\mathcal{AP}} = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$, where for $t \in \{call, ret, int\}$, $\Sigma_t = \{P \subseteq \mathcal{AP} \mid P \cap \{call, ret, int\} = \{t\}\}$.

A *Visibly Pushdown System* (VPS) over $\mathcal{AP}$ is a tuple $\mathcal{PS} = (\mathcal{AP}, Q = Q_{call} \cup Q_{ret} \cup Q_{int}, q_0, \Gamma \cup \{\bot\}, \mathit{Trans}, \mathit{Lab})$, where: (i) Q is a finite set of (control) states, which is partitioned into a set of call states Q_{call}, a set of return states Q_{ret}, and a set of internal states Q_{int}, (ii) $q_0 \in Q$ is the initial state, (iii) $\Gamma \cup \{\bot\}$ is a finite stack alphabet, (where $\bot \notin \Gamma$ is the special *stack bottom symbol*), (iv) $\mathit{Trans} \subseteq (Q_{call} \times Q \times \Gamma) \cup (Q_{ret} \times (\Gamma \cup \{\bot\}) \times Q) \cup (Q_{int} \times Q)$ is a transition relation, and (v) $\mathit{Lab} : Q \mapsto 2^{\mathcal{AP}}$ is a labelling function assigning to each control state $q \in Q$ the set $\mathit{Lab}(q)$ of propositions that hold over it such that for all $t \in \{call, ret, int\}$ and $q \in Q_t$, $\mathit{Lab}(q) \cap \{call, ret, int\} = \{t\}$.

Intuitively, from a call state $q \in Q_{call}$, $\mathcal{PS}$ chooses a push transition of the form $(q, q', \gamma) \in \mathit{Trans}$, pushes the symbol $\gamma \neq \bot$ onto the stack, and the control changes from q to q'. From a return state $q \in Q_{ret}$, $\mathcal{PS}$ chooses a pop transition of the form (q, γ, q'), where γ is popped from the stack (if $\gamma = \bot$, then γ is read but not popped). Finally, from an internal state $q \in Q_{int}$, $\mathcal{PS}$ can choose only transitions of the form (q, q') which do not use the stack.

A configuration of $\mathcal{PS}$ is a pair (q, β), where $q \in Q$ and $\beta \in \Gamma^* \cdot \{\bot\}$ is a stack content. The *initial configuration* is $(q_0, \bot)$ (the stack is initially empty). The VPS $\mathcal{PS}$ induces an infinite-state Kripke structure $\mathcal{K}_{\mathcal{PS}} = (\mathcal{AP}, S, E, \mathit{Lab}', s_0)$, where S is the set of configurations of $\mathcal{PS}$, s_0 is the initial configuration, and for all configurations $s = (q, \beta)$, $\mathit{Lab}'((q, \beta)) = \mathit{Lab}(q)$ and the set $E(s)$ of configurations s' such that $(s, s') \in E$ (*s-successors*) is defined as follows:

Push If $q \in Q_{call}$, then $E(s) = \{(q', \gamma \cdot \beta) \mid (q, q', \gamma) \in \mathit{Trans}\}$.

Pop If $q \in Q_{ret}$, then *either* $\beta = \bot$ and $E(s) = \{(q', \bot) \mid (q, \bot, q') \in \mathit{Trans}\}$, *or* $\beta = \gamma \cdot \beta'$, with $\gamma \in \Gamma$, and $E(s) = \{(q', \beta') \mid (q, \gamma, q') \in \mathit{Trans}\}$.

Internal If $q \in Q_{int}$, then $E(s) = \{(q', \beta) \mid (q, q') \in \mathit{Trans}\}$.

Note that the traces of $\mathcal{K}_{\mathcal{PS}}$ are words over the pushdown alphabet $\Sigma_{\mathcal{AP}}$. An *(initial) computation of* $\mathcal{PS}$ is an (initial) path in $\mathcal{K}_{\mathcal{PS}}$.

Nested BHS. We now focus on model-checking VPS against BHS formulas over a set of propositions $\mathcal{AP} \supseteq \{call, ret, int\}$. To that purpose, we extend the state-based branching-time approach presented in Section 2 by augmenting the set of atomic propositions $\mathcal{AP}$ with the special *well-matching proposition*, denoted by p_{wm}, which is fulfilled by a path of a Kripke structure over $\mathcal{AP}$ iff the associated trace is a well-matched finite word over $\Sigma_{\mathcal{AP}}$.

▶ **Definition 3.** *Let $\mathcal{K} = (\mathcal{AP}, S, E, Lab, s_0)$ be a Kripke structure over $\mathcal{AP}$. The* generalized *AIM induced by $\mathcal{K}$ is the AIM over $\mathcal{AP} \cup \{p_{wm}\}$ given by $\mathcal{N}_{\mathcal{K}} = (\mathcal{AP} \cup \{p_{wm}\}, \mathbb{I}, B_{\mathbb{I}}, E_{\mathbb{I}}, Lab_{\mathbb{I}})$, where $\mathbb{I}$, $B_{\mathbb{I}}$, $E_{\mathbb{I}}$, $Lab_{\mathbb{I}}^{-1}(p)$ for $p \in \mathcal{AP}$ are defined as in Definition 2, and $Lab_{\mathbb{I}}^{-1}(p_{wm})$ is the set of* finite *paths π of $\mathcal{K}$ such that $Lab(\pi)$ is well-matched. For a BHS formula ψ over $\mathcal{AP} \cup \{p_{wm}\}$ and a path π of $\mathcal{K}$, we write $\mathcal{K}, \pi \models_n \psi$ to mean that $\mathcal{N}_{\mathcal{K}}, \pi \models \psi$. $\mathcal{K}$ is a* nested model *of ψ, denoted $\mathcal{K} \models_n \psi$, if $\mathcal{K}, \pi \models_n \psi$ for all initial paths of $\mathcal{K}$.*

A *nested* BHS formula over $\mathcal{AP}$ is a BHS formula over $\mathcal{AP} \cup \{p_{wm}\}$. The *visibly pushdown model checking* (*VPMC*) *problem* against nested BHS is the problem of checking, given a visibly pushdown system $\mathcal{PS}$ and a nested BHS formula ψ (both over $\mathcal{AP}$), if $\mathcal{K}_{\mathcal{PS}} \models_n \psi$ holds.

We also consider the so-called *linear-time* fragment of nested BHS (nested $\mathsf{BHS}_{\mathsf{lin}}$ for short) obtained by imposing that modalities $\langle\overline{\mathrm{B}}\rangle$ and $\langle\overline{\mathrm{E}}\rangle$ occur in the scope of the binding modality $\mathbb{B}$. In nested $\mathsf{BHS}_{\mathsf{lin}}$ formulas ψ, the valuation of ψ depends only on the trace of the given path and is independent of the underlying Kripke structure. Formally, for all paths π and π' of (possibly distinct) Kripke structures $\mathcal{K}$ and $\mathcal{K}'$ having the same trace, it holds that $\mathcal{K}, \pi \models_n \psi$ iff $\mathcal{K}', \pi' \models_n \psi$. Thus, given a nested $\mathsf{BHS}_{\mathsf{lin}}$ formula ψ and a non-empty word w over $\Sigma_{\mathcal{AP}}$, we write $w \models_n \psi$ to mean that $\mathcal{K}, \pi \models_n \psi$ for any Kripke structure $\mathcal{K}$ with labeling Lab and path π such that $Lab(\pi) = w$.

In the following, we give some examples of how to use nested BHS as a specification language. For this, we introduce some auxiliary formulas which will be used as macro to specify more complex requirements. The formula $len_1 := [E]\,\texttt{false}$ captures the singular intervals (i.e. paths of length 1), and for a nested BHS formula ψ, the formulas $\mathit{left}(\psi) := \langle\overline{\mathrm{A}}\rangle(len_1 \wedge \psi)$ and $\mathit{right}(\psi) := \langle\mathrm{A}\rangle(len_1 \wedge \psi)$ assert that ψ holds at the singular intervals corresponding to the left and right endpoints, respectively, of the current finite interval. The formula $\theta_{mwm} := \mathit{left}(call) \wedge \mathit{right}(ret) \wedge p_{wm} \wedge [B]\neg p_{wm}$ characterizes the finite intervals whose first position is a matched call and the last position is the associated matching return, while the formula $\theta_{pc} := \xi_{ret} \wedge [B]\xi_{ret}$, where $\xi_{ret} := \mathit{right}(ret) \rightarrow (len_1 \vee \theta_{mwm} \vee \langle\mathrm{E}\rangle\,\theta_{mwm})$, captures intervals such that each non-first return position has a matched-call, i.e., fragments of computations π starting at a configuration s which precede the end (if any) of the procedural context associated with s. Finally, the formula $\theta_{loc} := \mathit{right}(\texttt{true}) \wedge \theta_{pc} \wedge \xi_{call} \wedge [E]\xi_{call}$, where $\xi_{call} := \mathit{left}(call) \rightarrow (len_1 \vee \theta_{mwm} \vee \langle\mathrm{B}\rangle\,\theta_{mwm})$, characterizes the finite intervals π satisfying θ_{pc} such that each non-last call position has a matching-return, i.e., the finite intervals π s.t. the first and last positions of π belong to the same *local procedural path* (alias *abstract path*). An abstract path captures the local computation within a procedure with the removal of subcomputations corresponding to nested procedure calls.

Specifying requirements. As we will show in Theorem 4, nested BHS strictly subsumes well-known context-free linear-time extensions of standard LTL, such as the logic CaRet [4]. In the analysis of recursive programs, an important feature of CaRet is that it allows to express in a natural way LTL requirements over two kinds of non-regular patterns on words over a pushdown alphabet: abstract paths and *caller paths* (a caller path represents the call-stack content at a given position). We show that CaRet formulas can be translated in polynomial time into nested BHS formulas of the form $\mathbb{B}\psi$ such that ψ is a nested HS formula (see

Theorem 4). It is worth noting that while CaRet provides ad hoc modalities for expressing abstract and caller properties, in nested BHS, we just use the special proposition p_{wm} and the regular modalities in BHS for expressing such non-regular context-free requirements. Additionally, nested BHS supports branching-time both in the past and in the future. In particular, the novel logic allows to specify in a natural way *procedural-context* (resp. abstract, resp. caller) versions of standard CTL and CTL* requirements which cannot be expressed in CaRet. As a first example, the *procedural-context version* of the CTL formula $\mathsf{E}(p_1 \mathsf{U} p_2)$, requiring that there is a computation π from the current configuration s such that the LTL formula $p_1 \mathsf{U} p_2$ holds along a prefix of π which precedes the end (if any) of the procedural context associated with s, can be expressed in nested HS by $\langle \mathrm{A} \rangle(\theta_{pc} \wedge [B]p_1 \wedge \mathit{right}(p_2))$, where $\langle \mathrm{A} \rangle$ plays the role of the existential path quantifier E of CTL*. Similarly, the *abstract version* of $\mathsf{E}(p_1 \mathsf{U} p_2)$, requiring that there is an abstract path from the current configuration s such that the LTL formula $p_1 \mathsf{U} p_2$ holds, can be expressed by $\langle \mathrm{A} \rangle\{\theta_{loc} \wedge \mathit{right}(p_2) \wedge [B](\theta_{loc} \rightarrow (\mathit{left}(p_1) \wedge \mathit{right}(p_1)))\}$.

As another example, we consider a *generalized version* of the total correctness requirement for a procedure A (popular in formalisms like Hoare logic), requiring that if a precondition *pre* is satisfied when A is called and an additional condition p eventually holds at a configuration s preceding the return (if any) of procedure A, then there is a computation from s such that A terminates and the post condition *post* holds upon return. This requirement can be expressed by the following nested HS formula, where c_A denotes invocation of procedure A: $[E]\{(\mathit{left}(\mathit{call} \wedge c_A \wedge \mathit{pre}) \wedge \mathit{right}(p) \wedge \theta_{pc}) \rightarrow \langle \overline{\mathrm{B}} \rangle(\theta_{mwm} \wedge \mathit{right}(\mathit{post}))\}$. Note that for expressing the previous branching-time requirement, we cannot simply use the existential path quantifier of CTL* corresponding to $\langle \mathrm{A} \rangle$, but we need to keep track of the current interval satisfying θ_{pc}, and we exploit modality $\langle \overline{\mathrm{B}} \rangle$ to nondeterministically extend this interval in the future.

We now consider the ability of expressing past branching-time modalities. Assume that the initial state q_0 is characterized by proposition *init*, q_0 is not a return state, and q_0 is not strictly reachable by any state. Then, the requirement that for every reachable configuration s where procedure A is called, s can be also reached in such a way that procedure B is on the call-stack can be expressed in nested HS by the formula $\mathit{right}(\mathit{call} \wedge c_A) \rightarrow \langle \overline{\mathrm{E}} \rangle\{\mathit{left}(\mathit{call} \wedge c_B) \wedge \theta_{pc} \wedge (\mathit{left}(\mathit{init}) \vee \langle \overline{\mathrm{E}} \rangle(\mathit{left}(\mathit{init}) \wedge \theta_{pc}))\}$.

As a last example, we consider the ability of specifying different properties at different returns of a procedural call depending on the behavior of the different branches in the called procedural context. For instance, let us consider the requirement that whenever a procedure is invoked, there are at least two branches in the called procedural context which return and: in one of them, condition p eventually holds and condition q holds upon the return, while in the other one, p never holds and q does not hold upon the return. This can be expressed by $\mathit{right}(\mathit{call}) \rightarrow \{\langle \mathrm{A} \rangle(\theta_{mwm} \wedge \mathit{right}(q) \wedge \langle \mathrm{B} \rangle \langle \mathrm{E} \rangle p) \wedge \langle \mathrm{A} \rangle(\theta_{mwm} \wedge \mathit{right}(\neg q) \wedge \neg \langle \mathrm{B} \rangle \langle \mathrm{E} \rangle p)\}$.

Expressiveness issues for nested BHS. Given two logics $\mathcal{F}_1$ and $\mathcal{F}_2$ interpreted over Kripke structures on $\mathcal{AP} \supseteq \{\mathit{call}, \mathit{ret}, \mathit{int}\}$, and two formulas $\varphi_1 \in \mathcal{F}_1$ and $\varphi_2 \in \mathcal{F}_2$, we say that φ_1 *and* φ_2 *are equivalent* if φ_1 and φ_2 have the same Kripke structure models. We say that $\mathcal{F}_2$ *is subsumed by* $\mathcal{F}_1$, written $\mathcal{F}_1 \geq \mathcal{F}_2$, if for each formula $\varphi_2 \in \mathcal{F}_2$, there is a formula $\varphi_1 \in \mathcal{F}_1$ such that φ_1 and φ_2 are equivalent. Moreover, $\mathcal{F}_1$ is *as expressive as* (resp., *strictly more expressive than*) $\mathcal{F}_2$ if both $\mathcal{F}_1 \geq \mathcal{F}_2$ and $\mathcal{F}_2 \geq \mathcal{F}_1$ (resp., $\mathcal{F}_1 \geq \mathcal{F}_2$ and $\mathcal{F}_2 \not\geq \mathcal{F}_1$).

We compare nested $\mathsf{BHS}_{\mathsf{lin}}$ with known context-free linear-time extensions of LTL, namely CaRet [4], NWTL [2], and the extension of CaRet with the *within* modality W (see [2]). Recall that NWTL and CaRet + W are expressively complete for the known context-free extension FO_μ of standard first-order logic (FO) over words (on a pushdown alphabet) by a binary call/matching return predicate [2], while it is an open question whether the same holds for CaRet [2]. Our expressiveness results can be summarized as follows.

▶ **Theorem 4.**

1. *Nested* $\mathsf{BHS_{lin}}$ *has the same expressiveness as* FO_μ*, and* NWTL *(resp.,* $\mathsf{CaRet} + \mathsf{W}$*) can be translated in polynomial time into equivalent nested* $\mathsf{BHS_{lin}}$ *formulas* ψ*, where for* CaRet *formulas,* ψ *is of the form* $\mathbb{B}\psi'$ *for some nested* HS *formula* ψ'*.*
2. *Nested* BHS *is strictly more expressive than* FO_μ*.*
3. HS *(hence,* BHS *as well) is strictly more expressive than standard* CTL^**.*

Sketched proof. Due to lack of space, the proof of Statement 1 is omitted. Statement 2 easily follows from Statement 1 and the fact that nested BHS supports branching-time. For example, let us consider the classical branching-time requirement asserting that from each state reachable from the initial one, it is possible to reach a state where proposition p holds. It is well-known that this formula is not FO-definable (see [7], Theorem 6.21). Hence, it is not FO_μ-definable as well (on Kripke structures having labeling Lab such that $int \in Lab(s)$ for each state, FO and FO_μ are equivalent). On the other hand, the previous requirement can be easily expressed in HS. Now, let us consider Statement 3. In [10], it is shown that in the state-based setting and under the homogeneity principle adopted in this paper, but assuming that intervals are associated with only finite paths of the Kripke structure, it holds that HS is strictly more expressive than *finitary* CTL^* (a variant of standard CTL^* where path quantification ranges over finite paths). By allowing also infinite paths and trivially adapting the results in [10], we deduce that HS (as considered in this paper) is strictly more expressive than standard CTL^*. ◀

4 Decision procedures

In this section, we show that the *VPMC* problem against nested BHS is decidable. The proof is based on a non-trivial automata-theoretic approach consisting in translating a given nested BHS formula ψ into a *Non-deterministic Visibly Pushdown Automaton* (NVPA) [5] accepting encodings of the computations of the given VPS $\mathcal{PS}$ which satisfy the formula ψ.

Details about the syntax and semantics of NVPA can be found in [5]. Here, we consider NVPA equipped with two sets F and F_ω of accepting states: F is used for acceptance of finite words, and F_ω for acceptance of infinite words. For an NVPA $\mathcal{A}$, we denote by $L(\mathcal{A})$ the language of finite and infinite words over Σ accepted by $\mathcal{A}$ (*Visibly Pushdown Language*).

We fix a visibly pushdown system $\mathcal{PS} = (\mathcal{AP}, Q_{\mathcal{PS}}, q^0_{\mathcal{PS}}, \Gamma_{\mathcal{PS}} \cup \{\bot\}, \mathit{Trans}_{\mathcal{PS}}, \mathit{Lab}_{\mathcal{PS}})$ over $\mathcal{AP}$, where $Q_{\mathcal{PS}} = Q_{call} \cup Q_{ret} \cup Q_{int}$. For encoding computations of $\mathcal{PS}$, we adopt the pushdown alphabet $\Sigma_{\mathcal{PS}} = \Sigma_{call} \cup \Sigma_{ret} \cup \Sigma_{int}$ defined as follows: $\Sigma_{call} := Q_{call} \cup \Gamma_{\mathcal{PS}} \cup (Q_{call} \times \Gamma_{\mathcal{PS}})$, $\Sigma_{ret} := Q_{ret}$, and $\Sigma_{int} := Q_{int}$. Thus, the return (resp., internal) symbols in $\Sigma_{\mathcal{PS}}$ are the return (resp., internal) states of $\mathcal{PS}$, while the set of calls consists of the call states of $\mathcal{PS}$ together with the stack symbols, and the pairs call state/stack symbol. Given a finite word w over $\Sigma_{\mathcal{PS}} \setminus Q_{call}$, the *unmatched call part* $\mathsf{umc}(w)$ *of* w is the word over $\Gamma_{\mathcal{PS}}$ defined as follows: let $h_0 < \ldots < h_{n-1}$ be the (possibly empty) sequence of unmatched call positions of w, then $\mathsf{umc}(w) = \gamma_0 \ldots \gamma_{n-1}$, where for each $0 \leq i \leq n-1$, γ_i is the $\Gamma_{\mathcal{PS}}$-component of $w(h_i)$.

We encode the computations π of $\mathcal{PS}$ by words over $\Sigma_{\mathcal{PS}}$ consisting of a prefix (the *head*) over $\Gamma_{\mathcal{PS}}$ encoding the stack content of the first configuration of π, followed by a word over $\Sigma_{\mathcal{PS}} \setminus \Gamma_{\mathcal{PS}}$ (the *body*) which keeps track of the states visited by π together with the stack-top symbols pushed from the non-last configurations of π associated with the call states.

▶ **Definition 5** (Computation-codes). *A computation-code (of* $\mathcal{PS}$*) is a word* w *over the pushdown alphabet* $\Sigma_{\mathcal{PS}}$ *of the form* $w = w_h \cdot w_b$ *such that the prefix* w_h *(the* head*) is a word in* $\Gamma^*_{\mathcal{PS}}$ *and the suffix* w_b *(the* body*) is either a* non-empty *finite word in* $((Q_{call} \times \Gamma_{\mathcal{PS}}) \cup Q_{ret} \cup$

$Q_{int})^* \cdot (Q_{call} \cup Q_{ret} \cup Q_{int})$*, or an infinite word over* $(Q_{call} \times \Gamma_{\mathcal{PS}}) \cup Q_{ret} \cup Q_{int}$*. Moreover, the body* w_b *satisfies the following conditions for each* $0 \leq i < |w_b| - 1$ *(*$\infty - 1$ *is for* ∞*), where for a symbol* $\sigma \in \Sigma_{\mathcal{PS}} \setminus \Gamma_{\mathcal{PS}}$*, we denote by* $q(\sigma)$ *the* $Q_{\mathcal{PS}}$*-component of* σ*:*

- *if* $w_b(i)$ *is a call, then* $w_b(i) = (q, \gamma)$ *and* $(q, q(w_b(i+1)), \gamma) \in Trans_{\mathcal{PS}}$*.*
- *if* $w_b(i)$ *is an internal action then* $(w_b(i), q(w_b(i+1))) \in Trans_{\mathcal{PS}}$*.*
- *if* $w_b(i)$ *is a return, then* $(w_b(i), \gamma, q(w_b(i+1))) \in Trans_{\mathcal{PS}}$*, where* $\gamma = \bot$ *if the return position* $i + |w_h|$ *has no matched-call in* $w = w_h \cdot w_b$*; otherwise,* γ *is the* $\Gamma_{\mathcal{PS}}$*-component of* $w(i_c)$*, where* i_c *is the matched-call position of* $i + |w_h|$*.*

We denote by $\Pi_{\mathcal{PS}}$ the set of computation-codes. Clearly, there is a bijection between $\Pi_{\mathcal{PS}}$ and the set of computations of $\mathcal{PS}$. In particular, a computation code $w = w_h \cdot w_b$ encodes the computation π_w of $\mathcal{PS}$ of length $|w_b|$ given by $\pi_w := (q(w_b(0)), \beta_0 \cdot \bot)(q(w_b(1)), \beta_1 \cdot \bot) \ldots$, where for each $0 \leq i < |w_b|$, β_i is the reverse of the unmatched call part of the prefix of w until position $i + |w_h| - 1$. Note that the head w_h encodes the stack content of the first configuration, i.e., $\beta_0 = (w_h)^R \cdot \bot$, where $(w_h)^R$ is the reverse of w_h.

We now illustrate the translation of nested BHS formulas over $\mathcal{AP}$ into a subclass of NVPA over $\Sigma_{\mathcal{PS}}$, we call $\mathcal{PS}$-NVPA. A $\mathcal{PS}$-NVPA is simply an NVPA over $\Sigma_{\mathcal{PS}}$ accepting only computation-codes. The following result is straightforward.

▶ **Proposition 6.** *One can construct a* $\mathcal{PS}$*-NVPA* $\mathcal{A}_{\mathcal{PS}}$ *with* $O(|\Sigma_{\mathcal{PS}}|)$ *states and* $O(|\Gamma_{\mathcal{PS}}|)$ *stack symbols accepting the set* $\Pi_{\mathcal{PS}}$ *of computation-codes.*

For the Boolean connectives in nested BHS formulas, we exploit the well-known closure of NVPA under language Boolean operations [5]. In particular the following holds.

▶ **Proposition 7** (Closure under intersection and complementation [5]). *Given two* $\mathcal{PS}$*-NVPA* $\mathcal{A}$ *and* $\mathcal{A}'$ *with* n *and* n' *states, and* m *and* m' *stack symbols, respectively, one can construct (i) a* $\mathcal{PS}$*-NVPA with* $n \cdot n'$ *states and* $m \cdot m'$ *stack symbols accepting* $L(\mathcal{A}) \cap L(\mathcal{A}')$*, and (ii) a* $\mathcal{PS}$*-NVPA with* $O(|\Sigma_{\mathcal{PS}}| \cdot 2^{n^2})$ *states and* $O(|\Sigma_{\mathcal{PS}}|^2 \cdot 2^{n^2})$ *stack symbols accepting* $\Pi_{\mathcal{PS}} \setminus L(\mathcal{A})$*.*

We now extend in a natural way the semantics of the HS modalities $\langle \mathrm{B} \rangle$, $\langle \overline{\mathrm{B}} \rangle$, $\langle \mathrm{E} \rangle$, $\langle \overline{\mathrm{E}} \rangle$ to languages L of words over the pushdown alphabet $\Sigma_{\mathcal{PS}}$, i.e. we interpret the $\langle \mathrm{B} \rangle$, $\langle \overline{\mathrm{B}} \rangle$, $\langle \mathrm{E} \rangle$, $\langle \overline{\mathrm{E}} \rangle$ modalities as operators over languages on $\Sigma_{\mathcal{PS}}$. The translation of nested BHS formulas into $\mathcal{PS}$-NVPA is crucially based on the closure of $\mathcal{PS}$-NVPA under such language operations.

For a computation-code $w \in \Pi_{\mathcal{PS}}$ encoding a $\mathcal{PS}$-computation π_w, we denote by $\mathrm{Pref}_{\mathcal{PS}}(w)$ the set of computation-codes encoding the computations in $\mathrm{Pref}(\pi_w)$, and by $\mathrm{Suff}_{\mathcal{PS}}(w)$ the set of computation-codes encoding the computations in $\mathrm{Suff}(\pi_w)$. Given a language L over $\Sigma_{\mathcal{PS}}$, let $\langle \mathrm{B} \rangle_{\mathcal{PS}}(L)$, $\langle \mathrm{E} \rangle_{\mathcal{PS}}(L)$, $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}(L)$, $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}(L)$ be the languages over $\Sigma_{\mathcal{PS}}$ defined as follows:

- $\langle \mathrm{B} \rangle_{\mathcal{PS}}(L) = \{w \in \Pi_{\mathcal{PS}} \mid \mathrm{Pref}_{\mathcal{PS}}(w) \cap L \neq \emptyset\}$;
- $\langle \mathrm{E} \rangle_{\mathcal{PS}}(L) = \{w \in \Pi_{\mathcal{PS}} \mid \mathrm{Suff}_{\mathcal{PS}}(w) \cap L \neq \emptyset\}$;
- $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}(L) = \{w \in \Pi_{\mathcal{PS}} \mid \exists\, w' \in \Pi_{\mathcal{PS}} \cap L \text{ such that } w \in \mathrm{Pref}_{\mathcal{PS}}(w')\}$;
- $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}(L) = \{w \in \Pi_{\mathcal{PS}} \mid \exists\, w' \in \Pi_{\mathcal{PS}} \cap L \text{ such that } w \in \mathrm{Suff}_{\mathcal{PS}}(w')\}$.

We show that $\mathcal{PS}$-NVPA are closed under the above language operations. We start with the prefix operator $\langle \mathrm{B} \rangle_{\mathcal{PS}}$ and the suffix operator $\langle \mathrm{E} \rangle_{\mathcal{PS}}$.

▶ **Proposition 8** (Closure under $\langle \mathrm{B} \rangle_{\mathcal{PS}}$ and $\langle \mathrm{E} \rangle_{\mathcal{PS}}$). *Given a* $\mathcal{PS}$*-NVPA* $\mathcal{A}$ *with* n *states and* m *stack symbols, one can construct in polynomial time a* $\mathcal{PS}$*-NVPA with* $O(n \cdot |\Sigma_{\mathcal{PS}}|)$ *states and* $O(m \cdot |\Gamma_{\mathcal{PS}}|)$ *stack symbols accepting* $\langle \mathrm{B} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$ *(resp.,* $\langle \mathrm{E} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$*).*

Proof. Let us consider the suffix operator $\langle \mathrm{E} \rangle_{\mathcal{PS}}$ (the closure under $\langle \mathrm{B} \rangle_{\mathcal{PS}}$ is straightforward). Let $\mathcal{A}$ be a $\mathcal{PS}$-NVPA with set of states Q and stack alphabet Γ. We first construct an NVPA $\mathcal{A}'$ with $O(|Q|)$ states and $O(|\Gamma|)$ stack symbols accepting the set of words w over $\Sigma_{\mathcal{PS}}$ such that there is a non-empty proper prefix w' of w over $\Sigma_{\mathcal{PS}} \setminus Q_{call}$ so that the last symbol of w' is in $\Sigma_{\mathcal{PS}} \setminus \Gamma_{\mathcal{PS}}$ and $\mathsf{umc}(w') \cdot w''$ is accepted by $\mathcal{A}$, where w'' is the remaining portion of w, i.e. $w = w' \cdot w''$. Since $\mathcal{A}$ accepts only words in $\Pi_{\mathcal{PS}}$, our encoding ensures that the $\mathcal{PS}$-NVPA accepting $\langle \mathrm{E} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$ and satisfying the statement of the theorem is given by the synchronous product of $\mathcal{A}'$ with the $\mathcal{PS}$-NVPA $\mathcal{A}_{\mathcal{PS}}$ accepting $\Pi_{\mathcal{PS}}$ of Proposition 6.

We now illustrate the construction of the NVPA $\mathcal{A}'$. Intuitively, $\mathcal{A}'$ guesses a non-empty proper prefix w' of the given input w such that the last symbol of w' is in $\Sigma_{\mathcal{PS}} \setminus \Gamma_{\mathcal{PS}}$ and checks that there is an accepting run of $\mathcal{A}$ over $\mathsf{umc}(w') \cdot w''$, where $w = w' \cdot w''$. The behaviour of $\mathcal{A}'$ is split in two phases. In the first phase, starting from an initial state of $\mathcal{A}$, $\mathcal{A}'$ simulates the behaviour of $\mathcal{A}$ over the unmatched call part $\mathsf{umc}(w')$ of the guessed prefix w' of the input. In the second phase, $\mathcal{A}'$ simply simulates the behaviour of $\mathcal{A}$ over w'' and accepts if and only if $\mathcal{A}$ accepts. $\mathcal{A}'$ keeps track in its (control) state of the current state of the simulated run of $\mathcal{A}$ over $\mathsf{umc}(w') \cdot w''$. Whenever a call position i_c is read along the guessed prefix w', $\mathcal{A}'$ guesses that one of the following two conditions holds:

- *i_c is a matched-call position in the guessed prefix w':* $\mathcal{A}'$ pushes a special symbol # on the stack, and the Q-component of the state remains unchanged. Moreover, in order to ensure that the guess is correct, $\mathcal{A}'$ exploits a flag mc. Intuitively, the flag mc marks the current state iff the current input position has a caller whose matching return exists in the guessed prefix w'. The transition function of $\mathcal{A}'$ ensures that the flag is propagated consistently with the guesses. In particular, on reading a call of w' in a state marked by mc, the flag mc is pushed onto the stack in order to be recovered on reading the matching-return. The guesses are ensured to be correct by requiring that the second phase can start only if the flag mc does not appear in the current state.
- *i_c is an unmatched-call position in the guessed prefix w':* from the current state with Q-component q, $\mathcal{A}'$ guesses a push-transition $q \xrightarrow{c,push(\gamma)} q'$ of $\mathcal{A}$ such that c is the $\Gamma_{\mathcal{PS}}$-component of $w'(i_c)$, pushes γ on the stack and moves to a state whose Q-component is q'. $\mathcal{A}'$ ensures that in the first phase no symbol in Γ can be popped from the stack.

Note that in the first phase, on reading a non-call position, the Q-component of the state of $\mathcal{A}'$ remains unchanged. ◀

Next, we consider the prefix-converse operator $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}$ and the suffix-converse operator $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}$. For an NVPA $\mathcal{A}$ and state q, $\mathcal{A}^q$ denotes the NVPA defined as $\mathcal{A}$ but with set of initial states given by $\{q\}$. Given states q and p of $\mathcal{A}$, a *summary of $\mathcal{A}$ from q to p* is a run of $\mathcal{A}^q$ over some finite well-matched word leading to a configuration whose associated state is p. A *minimally well-matched word* is a non-empty finite well-matched word w whose first position is a call having as matching-return the last position of w. We denote by $MR(\Sigma_{\mathcal{PS}})$ the set of words over $\Sigma_{\mathcal{PS}}$ such that each return position has a matching call.

▶ **Proposition 9** (Closure under $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}$ and $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}$). *Given a $\mathcal{PS}$-NVPA $\mathcal{A}$ with n states and m stack symbols, one can construct in polynomial time*

1. *a $\mathcal{PS}$-NVPA with $O(n^2)$ states and $O(n \cdot m)$ stack symbols accepting $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$, and*
2. *a $\mathcal{PS}$-NVPA with $3n$ states and m stack symbols accepting $\langle \overline{\mathrm{E}} \rangle_{\mathcal{PS}}(L(\mathcal{A}))$.*

Proof. We focus on Property 1 (i.e. closure under $\langle \overline{\mathrm{B}} \rangle_{\mathcal{PS}}$). Let $\mathcal{A}$ be a $\mathcal{PS}$-NVPA with set of states Q and stack alphabet Γ. We first construct an NVPA $\mathcal{A}'$ over $\Sigma_{\mathcal{PS}}$ with $O(|Q|^2)$ states and $O(|Q||\Gamma|)$ stack symbols accepting the language $\langle \overline{\mathrm{B}} \rangle(L(\mathcal{A})) := \{w \in \Sigma_{\mathcal{PS}}^* \mid \exists\, w'' \in L(\mathcal{A})$ such that $w \in (\mathrm{Pref}(w'') \cup \{\varepsilon\})\}$.

Starting from $\mathcal{A}'$, one can trivially construct in linear time an NVPA $\mathcal{A}''$ over $\Sigma_{\mathscr{PS}}$ accepting the set of non-empty finite words v such that the last symbol of v is not in $\Gamma_{\mathscr{PS}}$ and the word obtained from v by replacing the last symbol of v with its $Q_{\mathscr{PS}}$-component is accepted by $\mathcal{A}'$. Since $\mathcal{A}$ accepts only words in $\Pi_{\mathscr{PS}}$, our encoding ensures that $\mathcal{A}''$ is a $\mathscr{PS}$-NVPA accepting $\langle\overline{\mathrm{B}}\rangle_{\mathscr{PS}}(L(\mathcal{A}))$, and the result follows. We describe now the construction of the NVPA $\mathcal{A}'$ accepting $\langle\overline{\mathrm{B}}\rangle(L(\mathcal{A}))$. Intuitively, given an input $w \in \Sigma^*_{\mathscr{PS}}$, $\mathcal{A}'$ guesses a right-extension $w \cdot w'$ of w with $w' \neq \varepsilon$ and checks that there is an accepting run of $\mathcal{A}$ over $w \cdot w'$. $\mathcal{A}'$ simulates the behaviour of $\mathcal{A}$ on the given input w. Additionally, whenever a call position i_c occurs, $\mathcal{A}'$ guesses that one of the following conditions holds:

- *i_c is a matched-call position of the input w:* in order to ensure that the guess is correct, as in the proof of Proposition 8, $\mathcal{A}'$ carries in its state a flag mc that marks the current state if the current input position has a caller whose matching return exists. The transition function of $\mathcal{A}'$ ensures that the flag is propagated consistently with the guesses and the acceptance condition on finite words ensures that the guesses are correct.
- *i_c is an unmatched call position both in the input w and in the guessed right-extension $w \cdot w'$:* in this case, $\mathcal{A}'$ pushes the special symbol *bad* on the stack (the transition function ensures that *bad* is never popped from the stack), and carries in the control state, by means of an additional flag uc, the information that in the guessed right-extension $w \cdot w'$, there are no unmatched return positions in w'.
- *i_c is an unmatched call position in the input w but has matching return i_r in the guessed right-extension $w \cdot w'$:* in this case, $\mathcal{A}'$ simulates the behavior of $\mathcal{A}$ by choosing from the current state q a push transition $q \xrightarrow{w(i_c),push(\gamma)} q'$ of $\mathcal{A}$, and, additionally, guesses a matching pop-transition $p \xrightarrow{r,push(\gamma)} p'$ of $\mathcal{A}$ associated with the guessed return position i_r. Then, $\mathcal{A}'$ pushes the special symbol *bad* on the stack and moves to a state which keeps track both of the next state q' in the simulated run of $\mathcal{A}$ over $w \cdot w'$ and the state p (we call *summary state*) associated with the guessed matching return position i_r. Additionally, if i_c is the first guessed unmatched call position with matched return in $w \cdot w'$ (i.e., the infix from i_c to i_r is the maximal minimally well-matched word containing the last position of the input w), $\mathcal{A}'$ chooses the matching pop-transition $p \xrightarrow{r,push(\gamma)} p'$ in such a way that for the target state p', $(L(\mathcal{A}^{p'}) \setminus \{\varepsilon\}) \neq \emptyset$ if the flag uc does not mark the current state (i.e., every call in w has a matching return in $w \cdot w'$), and $(L(\mathcal{A}^{p'}) \setminus \{\varepsilon\}) \cap MR(\Sigma_{\mathscr{PS}}) \neq \emptyset$ otherwise (w' has no unmatched return positions). By using the stack, the summary state p is propagated along the maximal abstract path of w from position $i_c + 1$. Whenever a new call $\hat{i}_c$ occurs, then either $\hat{i}_c$ has a matched return in the input w, or a matching return $\hat{i}_r$ in the guessed right-extension $w \cdot w'$. In the first case, $\mathcal{A}'$ pushes the summary state p onto the stack to recover it on reading the matching return. In the second case, $\mathcal{A}'$ chooses from the current state $\hat{q}$ a push transition $\hat{q} \xrightarrow{w(\hat{i}_c),push(\gamma)} \hat{q}'$ of $\mathcal{A}$ and a matching pop-transition $\hat{p} \xrightarrow{r,push(\gamma)} \hat{p}'$ of $\mathcal{A}$ associated with the return position $\hat{i}_r$ such that there exists a summary of $\mathcal{A}$ from state $\hat{p}'$ to the summary state p (such a summary corresponds to the portion of the guessed run of $\mathcal{A}$ over $w \cdot w'$ associated with the infix from position $\hat{i}_r + 1$ to position $i_r - 1$). Then, $\mathcal{A}'$ pushes the special symbol *bad* on the stack and moves to a state which keeps track both of the next state $\hat{q}'$ (*main state*) in the simulated run of $\mathcal{A}$ and the new summary state $\hat{p}$. When the input w is read, $\mathcal{A}'$ accepts only if there is summary of $\mathcal{A}$ from the current main state to the current summary state.

In case $\mathcal{A}'$ guesses that every call in the input w is either matched in w, or unmatched in the guessed right-extension $w \cdot w'$, then $\mathcal{A}'$ accepts only if for the final main state q, $(L(\mathcal{A}^q) \setminus \{\varepsilon\}) \neq \emptyset$ if no call has been guessed unmatched, and $(L(\mathcal{A}^q) \setminus \{\varepsilon\}) \cap MR(\Sigma) \neq \emptyset$

otherwise. By standard results, given states p and q of $\mathcal{A}$, checking whether there is a summary from p to q (resp., $(L(\mathcal{A}^q) \setminus \{\varepsilon\}) \neq \emptyset$, resp., $(L(\mathcal{A}^q) \setminus \{\varepsilon\}) \cap MR(\Sigma_{\mathcal{PS}}) \neq \emptyset$) can be done in polynomial time. Hence, $\mathcal{A}'$ can be constructed in polynomial time. ◀

We can now establish the main result of this paper.

▶ **Theorem 10.** *Given a VPS $\mathcal{PS}$ and a nested BHS formula ψ, one can construct a $\mathcal{PS}$-NVPA accepting the words encoding the computations π of $\mathcal{PS}$ s.t. $\mathcal{K}_{\mathcal{PS}}, \pi \models \psi$. Moreover, the VPMC problem for nested BHS (resp., nested $\mathsf{BHS}_{\mathsf{lin}}$) is decidable with a non-elementary complexity.*

Sketch of proof. We can easily show that nested $\mathsf{BHS}_{\mathsf{lin}}$ can be translated in linear-time into FO_μ. Hence, by [6], given a nested $\mathsf{BHS}_{\mathsf{lin}}$ formula θ, one can construct an NVPA $\mathcal{A}$ of size non-elementary in the size of θ accepting the words v over $\Sigma_{\mathcal{AP}}$ such that $v \models_n \theta$. Starting from $\mathcal{A}$, one can easily construct a $\mathcal{PS}$-NVPA $\mathcal{A}'$ accepting the set of words over $\Sigma_{\mathcal{PS}}$ encoding the computations π of $\mathcal{PS}$ such that $\mathcal{K}_{\mathcal{PS}}, \pi \models_n \theta$. Hence, the first part of the theorem holds for nested $\mathsf{BHS}_{\mathsf{lin}}$. Thus, since an arbitrary nested BHS formula can be seen as a nested HS formula whose atomic formulas are nested $\mathsf{BHS}_{\mathsf{lin}}$ formulas, and being non-emptiness of NVPA solvable in polynomial time, the first part of the theorem and non-elementary decidability of the considered problem easily follow from the result for nested $\mathsf{BHS}_{\mathsf{lin}}$ and Propositions 7–9. For the non-elementary lower-bound, we show that the result already holds for finite model-checking against $\mathsf{BHS}_{\mathsf{lin}}$. The proof is by a polynomial-time reduction from the universality problem for star-free regular expressions built from union, concatenation, and negation. This problem is known to have a non-elementary complexity [27]. ◀

5 Concluding remarks

We have introduced and proved decidable a branching-time context-free logical framework for visibly pushdown model-checking, based on an extension of standard HS under the state-based semantics over Kripke structures and the homogeneity assumption. Future work will focus on the problem of determining the exact complexity of the *VPMC* problem for nested HS and its relevant fragments, and the complexity for nested BHS in terms of the nesting depth of the binding modality. Another intriguing problem concerns the expressiveness of the binding modality: in particular, is (nested) BHS more expressive than (nested) HS? We are also motivated to study suitable generalizations of the homogeneity assumption about the behavior of proposition letters over intervals. Finally, an interesting issue concerns the expressiveness comparison of nested BHS and VP-μ [3].

References

1 J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-Order and Temporal Logics for Nested Words. In *Proc. 22nd LICS*, pages 151–160. IEEE Computer Society, 2007.

3 R. Alur, S. Chaudhuri, and P. Madhusudan. A fixpoint calculus for local and global program flows. In *Proc. 33rd POPL*, pages 153–165. ACM, 2006.

4 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.

5 R. Alur and P. Madhusudan. Visibly Pushdown Languages. In *Proc. 36th STOC*, pages 202–211. ACM, 2004.

6 R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of ACM*, 56(3):16:1–16:43, 2009.

7 C. Baier and J.P. Katoen. *Principles of Model Checking.* The MIT Press, 2008.

8 L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. An in-Depth Investigation of Interval Temporal Logic Model Checking with Regular Expressions. In *Proc. 15th SEFM*, LNCS 10469, pages 104–119. Springer, 2017.

9 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Model checking for fragments of the interval temporal logic HS at the low levels of the polynomial time hierarchy. *Inf. Comput.*, 262(Part):241–264, 2018.

10 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval vs. Point Temporal Logic Model Checking: An Expressiveness Comparison. *ACM Trans. Comput. Logic*, 20(1):4:1–4:31, 2019.

11 L. Bozzelli and C. Sánchez. Visibly Linear Temporal Logic. *J. Autom. Reasoning*, 60(2):177–220, 2018.

12 D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theor. Comput. Sci.*, 560:269–291, 2014.

13 D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Decidability and Complexity of the Fragments of the Modal Logic of Allen's Relations over the Rationals. *Information and Computation*, accepted for publication on February 20, 2019.

14 K. Chatterjee, D. Ma, R. Majumdar, T. Zhao, T.A. Henzinger, and J. Palsberg. Stack Size Analysis for Interrupt-Driven Programs. In *Proc. 10th SAS*, LNCS 2694, pages 109–126. Springer, 2003.

15 T. Chen, F. Song, and Z. Wu. Global Model Checking on Pushdown Multi-Agent Systems. In *Proc. 30th AAAI*, pages 2459–2465. AAAI Press, 2016.

16 E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

17 J. Y. Halpern and Y. Shoham. A Propositional Modal Logic of Time Intervals. *Journal of the ACM*, 38(4):935–962, 1991.

18 A. Lomuscio and J. Michaliszyn. An Epistemic Halpern-Shoham Logic. In *Proc. 23rd IJCAI*, pages 1010–1016, 2013.

19 A. Lomuscio and J. Michaliszyn. Decidability of model checking multi-agent systems against a class of EHS specifications. In *Proc. 21st ECAI*, pages 543–548, 2014.

20 A. Lomuscio and J. Michaliszyn. Model Checking Multi-Agent Systems against Epistemic HS Specifications with Regular Expressions. In *Proc. 15th KR*, pages 298–308. AAAI Press, 2016.

21 A. Molinari, A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking interval properties of computations. *Acta Informatica*, 53(6-8):587–619, 2016.

22 A. Molinari, A. Montanari, and A. Peron. Model checking for fragments of Halpern and Shoham's interval temporal logic based on track representatives. *Inf. Comput.*, 259(3):412–443, 2018.

23 A. Montanari, G. Puppis, and P. Sala. A decidable weakening of Compass Logic based on cone-shaped cardinal directions. *Logical Methods in Computer Science*, 11(4), 2015.

24 B. Moszkowski. *Reasoning About Digital Circuits.* PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1983.

25 A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57. IEEE, 1977.

26 I. Pratt-Hartmann. Temporal propositions and their logic. *Artificial Intelligence*, 166(1-2):1–36, 2005.

27 L. J. Stockmeyer. The complexity of decision problems in automata theory and logic. *PhD thesis, MIT*, 1974.

28 Y. Venema. Expressiveness and Completeness of an Interval Tense Logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990.

29 I. Walukiewicz. Pushdown Processes: Games and Model Checking. In *Proc. 8th CAV*, pages 62–74, 1996.

Taming the Complexity of Timeline-Based Planning over Dense Temporal Domains

Laura Bozzelli
University of Napoli "Federico II", Napoli, Italy

Angelo Montanari
University of Udine, Udine, Italy

Adriano Peron
University of Napoli "Federico II", Napoli, Italy

Abstract

The problem of timeline-based planning (TP) over dense temporal domains is known to be undecidable. In this paper, we introduce two semantic variants of TP, called *strong minimal and weak minimal* semantics, which allow to express meaningful properties. Both semantics are based on the minimality in the time distances of the existentially-quantified time events from the universally-quantified reference event, but the weak minimal variant distinguishes minimality in the past from minimality in the future. Surprisingly, we show that, despite the (apparently) small difference in the two semantics, for the strong minimal one, the TP problem is still undecidable, while for the weak minimal one, the TP problem is just **PSPACE**-complete. Membership in **PSPACE** is determined by exploiting a strictly more expressive extension (ECA^+) of the well-known robust class of Event-Clock Automata (ECA) that allows to encode the weak minimal TP problem and to reduce it to non-emptiness of Timed Automata (TA). Finally, an extension of ECA^+(ECA^{++}) is considered, proving that its non-emptiness problem is undecidable. We believe that the two extensions of ECA (ECA^+ and ECA^{++}), introduced for technical reasons, are actually valuable per sé in the field of TA.

2012 ACM Subject Classification Computing methodologies → Planning under uncertainty; Theory of computation → Quantitative automata

Keywords and phrases Timeline-based planning, timed automata, event-clock automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.34

1 Introduction

Timeline-based planning (TP for short) is a promising approach to real-time temporal planning and reasoning about executions under uncertainty [10, 11, 13, 14, 15, 16]. Compared to classical action-based temporal planning [17, 28], TP adopts a more declarative paradigm which focuses on the constraints that sequences of actions have to fulfil to reach a given goal. In TP, the planning domain is modeled as a set of independent, but interacting, components, each one identified by a *state variable*. The temporal behaviour of a single state variable (component) is described by a sequence of *tokens* (*timeline*) where each token specifies a value of the variable (state) and the period of time during which it takes that value. The overall temporal behaviour (set of timelines) is constrained by a set of *synchronization rules* that specify quantitative temporal requirements between the time events (start-time and end-time) of distinct tokens. Synchronization rules have a very simple format: either *trigger rules*, expressing invariants and response properties (for each token with a fixed state, called *trigger*, there exist some tokens satisfying some mutual temporal relations), or *trigger-less ones*, expressing goals (there exist some tokens satisfying some mutual temporal relations). Notice that the way in which requirements are specified by synchronization rules corresponds to the "freeze" mechanism in the well-known timed temporal logic TPTL [1], which uses the freeze quantifier to bind a variable to a specific temporal context (a token in the TP setting).

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 34; pp. 34:1–34:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

TP has been successfully exploited in a number of application domains, including space missions, constraint solving, and activity scheduling (see, e.g., [4, 9, 12, 18, 23, 25]). A systematic study of expressiveness and complexity of TP has been undertaken only very recently in both the discrete-time and the dense-time settings [5, 6, 20, 21].

In the discrete-time case, the TP problem is **EXPSPACE**-complete, and expressive enough to capture action-based temporal planning (see [20, 21]). Despite the simple format of synchronization rules, the shift to a dense-time domain dramatically increases expressiveness and complexity, depicting a scenario which resembles that of the well-known timed linear temporal logics MTL and TPTL, under a pointwise semantics, which are undecidable in the general setting [1, 26]. The TP problem in its full generality is indeed undecidable [6], and undecidability is caused by the high expressiveness of trigger rules (if only trigger-less rules are used, the TP problem is just **NP**-complete [8]). Decidability can be recovered by imposing suitable syntactic/semantic restrictions on the trigger rules. In particular, two restrictions are considered in [5]: (i) the first one limits the comparison to tokens whose start times follow the start time of the trigger (*future semantics of trigger rules*); (ii) the second one imposes that a non-trigger token can be referenced at most once in the timed constraints of a trigger rule (*simple trigger rules*). Under these two restrictions, the TP problem is decidable with a non-primitive recursive complexity [5] and can be solved by a reduction to model checking of Timed Automata (TA) [2] against MTL over *finite* timed words, the latter being a known decidable problem [27]. By removing either the future semantics or the simple trigger rule restrictions, the TP problem turns out to be undecidable [6, 7].

Our Contribution. In this paper, without imposing any syntactic restriction to the format of synchronization rules, we investigate an alternative semantics for the trigger rules in the dense-time setting, which turns out to be still quite expressive and relevant for practical applications, and has the main advantage of guaranteeing a reasonable computational complexity. In the standard semantics of trigger rules, if there are many occurrences of non-trigger tokens carrying the same specified value, say v, nothing forces the choice of a specific occurrence. For instance, if the trigger token represents a prompt and the v-valued token is a reaction to it, the chosen v-valued token is not guaranteed to be the first one after issuing the prompt. In a reactive context, one is in general interested in relating an issued prompt to the first response to it and not to an arbitrarily delayed one. A similar idea is exploited by Event-Clock Automata (ECA) [3], a well-known robust subclass of Timed Automata(TA) [2]. In ECA, each symbol a of the alphabet is associated with a *recorder* or *past clock*, recording (at the current time) the time elapsed since the last occurrence of a, and a *predicting* or *future clock*, measuring the time required for the next occurrence of a.

The alternative semantics of trigger rules is based on the minimality in the time distances of the start times of existentially quantified tokens in a trigger rule from the start time of the trigger token. In fact, the minimality constraint can be used to express two alternative semantics: the *weak minimal semantics*, which distinguishes minimality in the past, with respect to the trigger token, from minimality in the future, and the *strong minimal semantics*, which considers minimality over all the start times (both in the past and in the future). Surprisingly, this apparently small difference in the definitions of weak and strong minimal semantics leads to a dramatic difference in the complexity-theoretic characterization of the TP problem: while the TP problem under the *strong minimal semantics* is still undecidable, the TP problem under the *weak minimal semantics* turns out to be **PSPACE**-complete (which is the complexity of the emptiness problem for TA and ECA [2, 3]). **PSPACE** membership of the weak minimal TP problem is shown by a non-trivial exponential-time reduction to non-emptiness of TA. To handle the trigger rules under the weak minimal semantics, we exploit, as an intermediate step in the reduction, a strictly more expressive extension of ECA, called ECA$^+$. This novel extension of ECA is obtained by allowing a larger class of atomic event-clock constraints, namely, diagonal constraints between clocks of the same polarity (past or future) and *sum constraints* between clocks of *opposite polarity*. In [19], these atomic constraints are used in *event-zones* to obtain symbolic forward and backward analysis semi-algorithms for ECA, which are not guaranteed to terminate. We show that, similar to ECA, ECA$^+$ are closed under language Boolean operations and can be translated in exponential time into equivalent TA with an exponential number of control states, but a linear number of clocks. We also investigate an extension of ECA$^+$, called ECA^{++}, where the polarity requirements in the diagonal and sum constraints are relaxed, and we show that the nonemptiness problem for such a class of automata is undecidable. We believe that these two original extensions of ECA, namely, ECA$^+$ and ECA^{++}, are interesting per sé, as they shed new light on the landscape of event-clock and timed automata.

The paper is organized as follows. In Section 2, we recall the TP framework and we introduce the strong and weak minimal semantics. In Section 3, we prove that the TP problem under the strong minimal semantics is still undecidable. In Section 4, we introduce and address complexity and expressiveness issues for ECA$^+$ and ECA^{++}. Moreover, by exploiting the results for ECA$^+$, we prove **PSPACE**-completeness of the weak minimal TP problem. Conclusions provide an assessment of the work done and outline future research themes.

2 The TP Problem

In this section, we recall the TP framework as presented in [15, 20] and we introduce the strong and weak minimal semantics. In TP, domain knowledge is encoded by a set of state variables, whose behaviour over time is described by transition functions and constrained by synchronization rules. We fix the following notation. Let $\mathbb{N}$ be the set of natural numbers, $\mathbb{R}_+$ the set of non-negative real numbers, and *Intv* the set of intervals in $\mathbb{R}_+$ whose endpoints are in $\mathbb{N} \cup \{\infty\}$. Given a finite word w over some alphabet (or, equivalently, a finite sequence of symbols), $|w|$ denotes the length of w and for all $0 \leq i < |w|$, $w(i)$ is the i-th letter of w.

▶ **Definition 1.** *A* state variable *x is a triple $x = (V_x, T_x, D_x)$, where V_x is the* finite domain *of the variable x, $T_x : V_x \to 2^{V_x}$ is the* value transition function*, which maps each $v \in V_x$ to the (possibly empty) set of successor values, and $D_x : V_x \to Intv$ is the* constraint function *that maps each $v \in V_x$ to an interval.*

A *token* for a variable x is a pair (v, d) consisting of a value $v \in V_x$ and a duration $d \in \mathbb{R}_+$ such that $d \in D_x(v)$. For a token $t = (v, d)$, *value*(t) denotes the first component v of t. Intuitively, a token for x represents an interval of time where the state variable x takes value v. The behavior of the state variable x is specified by means of *timelines* which are non-empty sequences of tokens $\pi = (v_0, d_0) \ldots (v_n, d_n)$ consistent with the value transition function T_x, that is, such that $v_{i+1} \in T_x(v_i)$ for all $0 \leq i < n$. We associate to the i-th token ($0 \leq i \leq n$) of the timeline π two punctual events: (i) the *start point* whose timestamp (*start time*), denoted by $\mathsf{s}(\pi, i)$, is 0 if $i = 0$, and is given by $\sum_{h=0}^{i-1} d_h$ otherwise, and (ii) the *end point* whose timestamp (*end time*), denoted by $\mathsf{e}(\pi, i)$, is given by $\mathsf{e}(\pi, i) := \mathsf{s}(\pi, i) + d_i$.

Given a finite set SV of state variables, a *multi-timeline* of SV is a mapping Π assigning to each state variable $x \in SV$ a timeline for x.

▶ **Example 2.** Assume to have transactions (e.g database transactions) accessing a common shared resource A for read/write operations. The resource A can be unlocked (un_A), read_locked (r_l_A) or write_locked (w_l_A). A state variable $x_A = (V_A, T_A, D_A)$ with

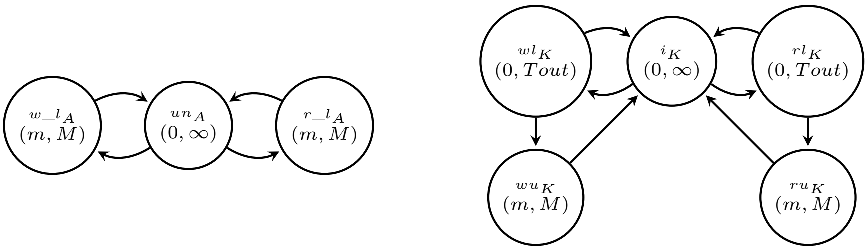


Figure 1 State variables x_A and x_K.

$V_A = \{un_A, r\_l_A, w\_l_A\}$ is used to describe the availability/locking of the resource during time. The value transition function T_A is represented as a graph in Figure 1 (left). Each node is labelled by a value v and by the constraint $D_A(v)$. The constants m and M are lower and upper bound, respectively, for the durations of read/write locking.
A state variable $x_K = (V_K, T_K, D_K)$, with K ranging over transaction names, describes the read/write locking requests issued by transaction K for the use of the resource A. A transaction can be idle (i_K), issuing a read or write lock for accessing the resource (rl_K or wl_K, resp.), or reading or writing the resource (ru_K or wu_K, resp.). Therefore we have $V_K = \{i_K, rl_K, wl_K, ru_K, wu_K\}$. An issued lock request can be accepted allowing the use of the resource or refused. There is a timeout $Tout$ for waiting the availability of the resource. The value transition and constraint functions T_K and D_K are depicted in Figure 1 (right).

Synchronization rules. Fix a finite set SV of state variables. Multi-timelines of SV can be constrained by a set of *synchronization rules*, which relate tokens, possibly belonging to different timelines, through temporal constraints on the start/end-times of tokens (point constraints) and on the difference between start/end-times of tokens (difference constraints). The synchronization rules exploit an alphabet Σ of token names to refer to the tokens along a multi-timeline, and are based on the notions of *atom* and *existential statement*.

An *atom* is either a clause of the form $ev(o) \in I$ (*point atom*), or of the form $ev(o) - ev'(o') \in I$ (*difference atom*), where $o, o' \in \Sigma$, $I \in Intv$, and $ev, ev' \in \{\mathsf{s}, \mathsf{e}\}$. Intuitively, an atom $ev(o) \in I$ asserts that the ev-time (i.e., the start-time if $ev = \mathsf{s}$, and the end-time otherwise) of the token referenced by o is in the interval I, while an atom $ev(o) - ev'(o') \in I$ requires that the difference between the ev-time and the ev'-time of the tokens referenced by o and o', respectively, is in I. Formally, an atom is evaluated with respect to a Σ-*assignment* λ_Π *for a given multi-timeline* Π *of* SV which is a mapping assigning to each token name $o \in \Sigma$ a pair $\lambda_\Pi(o) = (\pi, i)$ such that π is a timeline of Π and $0 \leq i < |\pi|$ (intuitively, (π, i) represents the token of Π referenced by the name o). An atom $ev(o) \in I$ (resp., $ev(o) - ev'(o') \in I$) *is satisfied by* λ_Π if $ev(\lambda_\Pi(o)) \in I$ (resp., $ev(\lambda_\Pi(o)) - ev'(\lambda_\Pi(o')) \in I$).

An *existential statement* $\mathcal{E}$ is a statement of the form $\mathcal{E} := \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n].\mathcal{C}$, where $\mathcal{C}$ is a conjunction of atoms, $o_i \in \Sigma$, $x_i \in SV$, and $v_i \in V_{x_i}$ for each $i = 1, \ldots, n$. The elements $o_i[x_i = v_i]$ are called *quantifiers*. A token name used in $\mathcal{C}$, but not occurring in any quantifier, is said to be *free*. Intuitively, the quantifier $o_i[x_i = v_i]$ binds the name o_i to some token in the timeline for variable x_i having value v_i. A Σ-assignment λ_Π for a multi-timeline Π of SV *satisfies* $\mathcal{E}$ if each atom in $\mathcal{C}$ is satisfied by λ_Π, and for each quantified token name o_i, $\lambda_\Pi(o_i) = (\pi, h)$ where $\pi = \Pi(x_i)$ and the h-th token of π has value v_i. A multi-timeline Π of SV *satisfies* $\mathcal{E}$ if there exists a Σ-assignment λ_Π for Π which satisfies $\mathcal{E}$.

▶ **Definition 3.** *A* synchronization rule $\mathcal{R}$ *for the set* SV *of state variables has the forms*
(trigger rule) $o_0[x_0 = v_0] \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$, *(trigger-less rule)* $\top \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$,
where $o_0 \in \Sigma$*,* $x_0 \in SV$*,* $v_0 \in V_{x_0}$*, and* $\mathcal{E}_1, \ldots, \mathcal{E}_k$ *are* existential statements*. In trigger rules, the quantifier* $o_0[x_0 = v_0]$ *is called* trigger*, and it is required that only* o_0 *may appear free in* $\mathcal{E}_i$ *(for* $i = 1, \ldots, k$*). For trigger-less rules, it is required that no token name appears free.*

Intuitively, a trigger $o_0[x_0 = v_0]$ acts as a universal quantifier, which states that *for all* the tokens of the timeline for the state variable x_0 having value v_0, at least one of the existential statements $\mathcal{E}_i$ must be true. Trigger-less rules simply assert the satisfaction of some existential statement. Formally, the *standard semantics* of the synchronization rules is defined as follows. A multi-timeline Π of SV *satisfies* a *trigger-less rule* $\mathcal{R}$ of SV if Π satisfies some existential statement of $\mathcal{R}$. Π *satisfies* a *trigger rule* $\mathcal{R}$ of SV with trigger $o_0[x_0 = v_0]$ if for every position i of the timeline $\Pi(x_0)$ for x_0 such that $\Pi(x_0)(i) = (v_0, d)$, there is an existential statement $\mathcal{E}$ of $\mathcal{R}$ and a Σ-assignment λ_Π for Π such that $\lambda_\Pi(o_0) = (\Pi(x_0), i)$ and λ_Π satisfies $\mathcal{E}$. Trigger-less are usually exploited to express initial conditions or the goals of the problem. Trigger rules are useful to specify invariants and response requirements.

▶ **Example 4.** With reference to Example 2, we introduce synchronization rules to guarantee that the shared resource A is accessed in mutual exclusion during writing. We first define some shorthand for expressing conjunctions of atoms where o and $\overline{o}$ are token names:

- $during(o, \overline{o}) := \mathsf{s}(\overline{o}) - \mathsf{s}(o) \in [0, \infty) \wedge \mathsf{e}(o) - \mathsf{e}(\overline{o}) \in [0, \infty)$ requires that the token referenced by $\overline{o}$ occurs during the token referenced by o;
- $overlap(o, \overline{o}) := \mathsf{e}(o) - \mathsf{s}(\overline{o}) \in (0, \infty) \wedge \bigwedge_{ev \in \{\mathsf{s}, \mathsf{e}\}} ev(\overline{o}) - ev(o) \in (0, \infty)$ asserts that the $\overline{o}$'s token does not start before the o's token and crosses the end point of the o's token.

The following first pair of trigger-less rules fix the initial conditions: the resource A is initially unlocked and each transaction K is idle. The next trigger rule ensures that when a transaction K reads the resource A, the resource is locked for reading. The last trigger rule requires that when K writes A, there is a write locking token of A having the same temporal window as the K-token.

- $\top \to \exists o[x_A = un_A].\, \mathsf{s}(o) \in [0, 0]$ and $\top \to \exists o[x_K = i_K].\, \mathsf{s}(o) \in [0, 0]$;
- $o_0[x_K = ru_K] \to \exists o[x_A = r\_l_A].during(o, o_0)$;
- $o_0[x_K = wu_K] \to \exists o[x_A = w\_l_A].during(o, o_0) \wedge during(o_0, o)$.

The two trigger rules above ensure the mutual exclusion among reads and writes of the resource A by the same transaction K. The following rules are added to guarantee mutual exclusion when distinct transactions K and H write A, i.e. we have to ensure that K and H do not feature tokens of value wu_K and wu_H, respectively, with the same temporal window.

$$\begin{aligned} o_0[x_K = wu_K] \to \; & \bigvee_{s \in \{i_H, wl_H, rl_H\}} \big(\exists o[x_H = s].during(o, o_0) \vee \exists o[x_H = s].during(o_0, o) \vee \\ & \exists o[x_H = s].overlap(o_0, o) \vee \exists o[x_H = s].overlap(o, o_0)\big). \end{aligned}$$

Minimal semantics of trigger rules. In the following we define the variant of the semantics for trigger rules newly proposed and investigated in this paper. It is obtained from the standard semantics by additionally requiring that the given Σ-assignment λ_Π selects for each (existential) quantifier $o[x = v]$ a token for variable x with value v whose start point has a minimal time distance from the start point of the trigger. Actually, the constraint of minimality can be used to express two alternative semantics: the *weak minimal semantics* which distinguishes minimality in the past (w.r.t. the trigger token) from the minimality in the future, and the *strong minimal semantics* which considers minimality over all the start times (both in the past and in the future) of the tokens for a variable x and x-value v.

▶ **Definition 5.** *Let $o_0 \in \Sigma$. A Σ-assignment λ_Π for a multi-timeline Π of SV is* weakly minimal *w.r.t. o_0 if for each $o \in \Sigma$ with $\lambda_\Pi(o) = (\pi, i)$, the following holds:*

- minimality in the past: *if* $\mathsf{s}(\pi, i) \leq \mathsf{s}(\lambda_\Pi(o_0))$ *then there is no position ℓ along the timeline π such that $value(\pi(i)) = value(\pi(\ell))$ and $\mathsf{s}(\pi, i) < \mathsf{s}(\pi, \ell) \leq \mathsf{s}(\lambda_\Pi(o_0))$.*
- minimality in the future: *if* $\mathsf{s}(\pi, i) \geq \mathsf{s}(\lambda_\Pi(o_0))$ *then there is no position ℓ along the timeline π such that $value(\pi(i)) = value(\pi(\ell))$ and $\mathsf{s}(\pi, i) > \mathsf{s}(\pi, \ell) \geq \mathsf{s}(\lambda_\Pi(o_0))$.*

A Σ-assignment λ_Π for Π is strongly minimal *w.r.t. o_0 if for each $o \in \Sigma$ with $\lambda_\Pi(o) = (\pi, i)$, there is no position ℓ along the timeline π such that $value(\pi(i)) = value(\pi(\ell))$ and $|\mathsf{s}(\pi, \ell) - \mathsf{s}(\lambda_\Pi(o_0))| < |\mathsf{s}(\pi, i) - \mathsf{s}(\lambda_\Pi(o_0))|$.*

The weak minimal (resp., strong minimal) semantics of the trigger rules is obtained from the standard one by imposing that the considered Σ-assignment λ_Π is weakly minimal (resp., strongly minimal) w.r.t. the trigger token o_0.

Note that we consider start points of tokens for expressing minimality. Equivalent semantics can be obtained by considering end points of tokens instead.

With reference to Example 4, we observe that, due to the constraints *during* and *overlap*, the weak minimal semantics for the considered trigger rules corresponds to the standard one.

Domains and plans. A TP domain $\mathcal{D} = (SV, R)$ is specified by a finite set SV of state variables and a finite set R of synchronization rules modeling their admissible behaviors. A *weak* (resp., *strong*) *minimal plan* of $\mathcal{D}$ is a multi-timeline of SV satisfying all the rules in R under the weak (resp., strong) minimal semantics of trigger rules. The *weak* (resp. *strong*) *minimal TP problem* is checking given a domain $\mathcal{D}$, whether there is a weak (resp. strong) minimal plan of $\mathcal{D}$. We also consider the *discrete-time versions* of the previous problems, where the durations of the tokens in a plan are restricted to be natural numbers.

▶ Assumption 6 (*Strict time monotonicity*). In the following, for simplifying the technical presentation of some results, without loss of generality, we assume that given a state variable $x = (V_x, T_x, D_x)$, the duration of a token for x is never zero, i.e., for each $v \in V_x$, $0 \notin D_x(v)$.

3 Undecidability of the strong minimal TP problem

In this section, we establish the negative result for the strong minimal semantics by a polynomial-time reduction from the *halting problem for Minsky* 2*-counter machines* [24]. The key feature in the reduction is the possibility to express for a given value v, a temporal equidistance requirement w.r.t. the start point of the trigger token for the start points of the last token before the trigger with value v and the first token after the trigger with value v.

▶ **Theorem 7.** *The strong minimal TP problem is undecidable even in the discrete-time setting.*

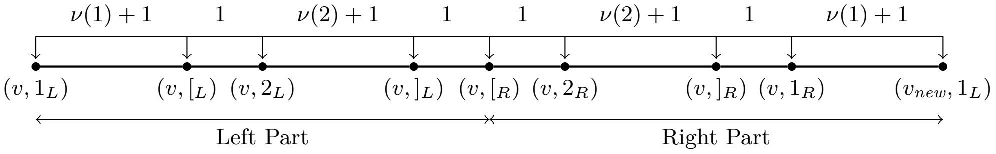
Proof. A nondeterministic Minsky 2-counter machine is a tuple $M = (Q, q_{init}, q_{halt}, \Delta)$, where Q is a finite set of (control) locations, $q_{init} \in Q$ is the initial location, $q_{halt} \in Q$ is the halting location, and $\Delta \subseteq Q \times L \times Q$ is a transition relation over the instruction set $L = \{\mathsf{inc}, \mathsf{dec}, \mathsf{zero\_test}\} \times \{1, 2\}$. For a transition $\delta = (q, op, q') \in \Delta$, we define $from(\delta) := q$, $op(\delta) := op$, and $to(\delta) := q'$. Without loss of generality we assume that:

- for each transition $\delta \in \Delta$, $from(\delta) \neq q_{halt}$ and $to(\delta) \neq q_{init}$, and
- there is exactly one transition in Δ, denoted δ_{init}, having as source location q_{init}.

An M-configuration is a pair (q, ν) consisting of a location $q \in Q$ and a counter valuation $\nu : \{1, 2\} \to \mathbb{N}$. A computation of M is a non-empty *finite* sequence $(q_1, \nu_1), \ldots, (q_k, \nu_k)$ of configurations such that for all $1 \leq i < k$, there is some instruction $op_i = (tag_i, c_i) \in L$, so that $(q_i, op_i, q_{i+1}) \in \Delta$ and: (i) $\nu_{i+1}(c) = \nu_i(c)$ if $c \neq c_i$; (ii) $\nu_{i+1}(c_i) = \nu_i(c_i)+1$ if $tag_i = \mathsf{inc}$; (iii) $\nu_{i+1}(c_i) = \nu_i(c_i) - 1$ and $\nu_i(c_i) > 0$ if $tag_i = \mathsf{dec}$; and (iv) $\nu_{i+1}(c_i) = \nu_i(c_i) = 0$ if $tag_i = \mathsf{zero\_test}$. The halting problem is to decide whether for a machine M, there is a computation starting at the *initial* configuration (q_{init}, ν_{init}), where $\nu_{init}(1) = \nu_{init}(2) = 0$, and leading to some halting configuration (q_{halt}, ν) (it was proved to be undecidable in [24]). To prove Theorem 7 we construct a TP instance $\mathcal{D}_M = (SV_M, R_M)$ such that M halts *iff* there exists a strong minimal discrete-time plan for $\mathcal{D}_M$.

We exploit a state variable x_M for encoding the evolution of the machine M and additional state variables for checking that the values of counters in the timeline for x_M are correctly updated. The domain V_M of the state variable x_M is given by $V_M := V_\Delta \times \{1_L, 1_R, 2_L, 2_R, [_L,]_L, [_R,]_R\}$ where V_Δ is the set of pairs $(\delta'_\perp, \delta)$, where $\delta'_\perp \in \Delta \cup \{\perp\}$, $\delta \in \Delta$, and $to(\delta'_\perp) = from(\delta)$ if $\delta'_\perp \neq \perp$, and $\delta = \delta_{init}$ otherwise. Intuitively, in the pair $(\delta'_\perp, \delta)$, δ represents the transition currently taken by M from the current non-halting configuration C, while $\delta'_\perp$ is $\perp$ if C is the initial configuration, and δ' represents the transition exploited by M in the previous computational step otherwise.

A configuration $C = (q, \nu)$ of M is encoded by the timelines π_C (*configuration codes*) of length 9 for the state variable x_M illustrated in the following figure, where $v \in V_\Delta$ (called *V_Δ-value of π_C*) is of the form $(\delta'_\perp, \delta)$ such that $from(\delta) = q$. Note that the configuration



code π_C is subdivided in two parts. In the left part (resp., right part), the encoding of counter 1 (resp., 2) precedes the encoding of counter 2 (resp., 1). The value $\nu(1)$ of counter 1 is encoded by the duration, which is $\nu(1) + 1$, of the *counter token* with value marked by 1_L in the left part, and the *counter token* with value marked by 1_R in the right part, and similarly for counter 2. The four tokens with values marked by $[_L$, $]_L$, $[_R$, and $]_R$, respectively, are called *tagged* tokens and their duration is always 1: they are used to check by trigger rules (under the strong minimal semantics) that increment and decrement M-instructions are correctly encoded. Moreover, we require that the configuration code π_C satisfies the following additional requirement (*V_Δ-requirement*), with $v = (\delta'_\perp, \delta)$ and $\delta = (q, op, q')$:

- $v_{new} = v$ if $to(\delta) = q_{halt}$, and v_{new} is of the form (δ, δ'') otherwise (*consecution*);
- if $\delta = \delta_{init}$ then the counter tokens have duration 1;
- if $op = (\mathsf{dec}, c)$ (resp., $op = (\mathsf{zero\_test}, c)$), then the durations of the counter tokens with values (v, c_L) and (v, c_R) are greater than 1 (resp., are equal to 1).

A *pseudo-configuration code* is defined as a configuration code but the durations of the counter tokens are arbitrary with the restriction that the V_Δ-requirement is fulfilled.

By construction and the assumption on M, we can easily define a *trigger-less* rule $\mathcal{R}_{init,halt}$ and define the transition and constraint function of x_M in such a way that the timelines of x_M satisfying $\mathcal{R}_{init,halt}$ (called *pseudo-computation codes*) are the sequences of the form $\pi_0 \cdots \pi_n$ such that: (i) $\pi_i \cdot \pi_{i+1}(0)$ and π_n are *pseudo-configuration codes* for all $0 \leq i < n$, (ii) if $n > 0$ (resp., $n = 0$), the V_Δ-value of $\pi_0 \cdot \pi_1(0)$ (resp., π_0) is $(\perp, \delta_{init})$ (*initialization*), and (iii) the V_Δ-value of π_n is of the form $(\delta'_\perp, \delta)$ such that $to(\delta) = q_{halt}$ (*halting*).

We now consider the crucial part of the reduction which has to guarantee that along a pseudo-computation code the counters are correctly encoded (i.e., the durations of the left and right tokens for each counter in a pseudo-configuration code coincide) and are updated accordingly to the M-instructions. Here, we focus on the increment instruction $(\mathsf{inc}, 1)$ for counter 1. For this, we exploit trigger rules in conjunction with an additional state variable $x_{(\mathsf{inc},1)}$ having domain $V_{check} := \{check_1, check_2, trigger, \bot\}$ and capturing the timelines π such that the duration of each token is at least 1 and the untimed part of π is an arbitrary non-empty word over V_{check}. Let π_M be a pseudo-computation code, π_C a non-initial pseudo-configuration code of π_M with V_Δ-value $(\delta'_\bot, \delta)$ such that $\delta'_\bot \neq \bot$ and $op(\delta'_\bot) = (\mathsf{inc}, 1)$ (we denote by $V_{(\mathsf{inc},1)}$ the set of such V_Δ-values), and π_{C_p} the pseudo-configuration code preceding π_C along π_M. We need to ensure that the duration of the token for counter 1 (resp., 2) in the left part of π_C is one plus the duration (resp., is the duration) of the token for counter 1 (resp., 2) in the right part of π_{C_p}. The proposed encoding ensures that the previous requirement holds *iff* for each token tk_{1_L} of π_M with value in $V_{(\mathsf{inc},1)} \times \{1_L\}$, the following holds ($(\mathsf{inc}, 1)$*-requirement*): for the last token marked by $]_R$ (resp., $[_R$) preceding tk_{1_L} and the first token marked by $[_L$ (resp., $]_L$) following tk_{1_L}, their start points have the same time distance from the start point of tk_{1_L}. Then, in order to enforce the $(\mathsf{inc}, 1)$-requirement, we first require that:

(*) the timelines π_M and $\pi_{(\mathsf{inc},1)}$ of variables x_M and $x_{(\mathsf{inc},1)}$, respectively, are synchronized, i.e., they have the same length and for each position i, the start-times of the ith tokens of π_M and $\pi_{(\mathsf{inc},1)}$ coincide;

(**) for the timeline $\pi_{(\mathsf{inc},1)}$ for variable $x_{(\mathsf{inc},1)}$ (synchronized with π_M), it holds that a token has value *trigger* (resp., has value $check_1$, resp., has value $check_2$) iff the associated token along π_M has a value in $V_{(\mathsf{inc},1)} \times \{1_L\}$ (resp., in $V_\Delta \times \{]_R, [_L\}$, resp., in $V_\Delta \times \{[_R,]_L\}$).

Since the duration of a token is not zero, the previous two requirements (*)–(**) can be easily expressed by trigger rules under the strong minimal semantics. Finally, we require that for each trigger-token $tk_{trigger}$ along the timeline $\pi_{(\mathsf{inc},1)}$ for $x_{(\mathsf{inc},1)}$ and for each $\ell = 1, 2$, the start points of the last $check_\ell$-token of $\pi_{(\mathsf{inc},1)}$ preceding $tk_{trigger}$ and the first $check_\ell$-token following $tk_{trigger}$ have the same time distance from the start point of $tk_{trigger}$. By the strong minimal semantics, this requirement can be expressed by the following two trigger rules, where $op = (\mathsf{inc}, 1)$, which ensure that for the $check_\ell$-tokens ($\ell = 1, 2$) whose start points have the smallest time distance from the start point of the trigger, there is one preceding the trigger and one following the trigger:

$$o[x_{op} = trigger] \to \exists o_1[x_{op} = check_1]\exists o_2[x_{op} = check_2]. \bigwedge_{\ell=1,2} \mathsf{s}(o) - \mathsf{s}(o_\ell) \in [0, \infty)$$
$$o[x_{op} = trigger] \to \exists o_1[x_{op} = check_1]\exists o_2[x_{op} = check_2]. \bigwedge_{\ell=1,2} \mathsf{s}(o_\ell) - \mathsf{s}(o) \in [0, \infty). \quad ◀$$

4 Decidability of the weak minimal TP problem

In this section, we show that the weak minimal TP problem is decidable and **PSPACE**-complete. The upper bound is obtained by an exponential-time reduction to nonemptiness of Timed Automata (TA) [2]. In order to handle the trigger rules under the weak minimal semantics, we exploit as an intermediate step an extension, denoted by ECA$^+$, of the known class of *Event Clock Automata* (ECA) [3]. The rest of the section is organized is follows. We first shortly recall the class of Timed Automata (TA) [2]. Then, in Subsection 4.1, we introduce and address complexity and expressiveness issues for the newly introduced class of ECA$^+$. Finally, in Subsection 4.2, we solve the weak minimal TP problem.

Let Σ be a finite alphabet. A *timed word* w over Σ is a *finite* word $w = (a_0, \tau_0) \cdots (a_n, \tau_n)$ over $\Sigma \times \mathbb{R}_+$ (τ_i is the time at which a_i occurs) such that $\tau_i \leq \tau_{i+1}$ for all $0 \leq i < n$ (monotonicity). The timed word w is also denoted by (σ, τ), where σ is the untimed word $a_0 \cdots a_n$ and $\tau = \tau_0 \cdots \tau_n$. A *timed language* over Σ is a set of timed words over Σ.

A TA over Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, C, \Delta, F)$, where Q is a finite set of (control) states, $Q_0 \subseteq Q$ is the set of initial states, C is a finite set of *clocks*, $F \subseteq Q$ is the set of accepting states, and Δ is the finite set of transitions (q, a, θ, Res, q') such that $q, q' \in Q$, $a \in \Sigma$, $Res \subseteq C$ is a clock reset set, and θ is a *clock constraint* over C, that is a conjunction of atomic formulas of the form $c \sim n$ (*simple constraints*) with $c \in C$, $\sim \in \{<, \leq, \geq, >\}$, and $n \in \mathbb{N}$. We denote by $K_{\mathcal{A}}$ the maximal constant used in the clock constraints of $\mathcal{A}$. Intuitively, in a TA $\mathcal{A}$, while transitions are instantaneous, time can elapse in a control state. The clocks progress at the same speed and can be reset independently of each other when a transition is executed, in such a way that each clock keeps track of the time elapsed since the last reset. Moreover, clock constraints are used as guards of transitions to restrict the behavior of the automaton. Formally, a configuration of $\mathcal{A}$ is a pair (q, val), where $q \in Q$ and $val : C \to \mathbb{R}_+$ is a clock valuation for C assigning to each clock a non-negative real number. For $t \in \mathbb{R}_+$ and a reset set $Res \subseteq C$, the valuations $(val + t)$ and $val[Res]$ are defined as: for all $c \in C$, $(val + t)(c) = val(c) + t$, and $val[Res](c) = 0$ if $c \in Res$ and $val[Res](c) = val(c)$ otherwise. For a clock constraint θ, val satisfies θ, written $val \models \theta$, if for each conjunct $c \sim n$ of θ, $val(c) \sim n$.

A run r of $\mathcal{A}$ on a timed word $w = (a_0, \tau_0) \cdots (a_n, \tau_n)$ over Σ is a sequence of configurations $r = (q_0, val_0) \cdots (q_{n+1}, val_{n+1})$ starting at an initial configuration (q_0, val_0), with $q_0 \in Q_0$ and $val_0(c) = 0$ for all $c \in C$, and such that for all $0 \leq i \leq n$ (we let $\tau_{-1} = 0$): $(q_i, a_i, \theta, Res, q_{i+1}) \in \Delta$ for some constraint θ and reset set Res, $(val_i + \tau_i - \tau_{i-1}) \models \theta$ and $val_{i+1} = (val_i + \tau_i - \tau_{i-1})[Res]$. The run r is *accepting* if $q_{n+1} \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ of $\mathcal{A}$ is the set of timed words w over Σ s.t. there is an accepting run of $\mathcal{A}$ over w.

4.1 Extended Event-clock Automata

In this section, we introduce an extension, denoted by ECA$^+$, of *Event Clock Automata* (ECA) [3]. In ECA, clocks have a predefined association with the input alphabet symbols and their values refer to the time distances from previous and next occurrences of input symbols. ECA$^+$ extend ECA by allowing a larger class of atomic event-clock constraints, namely diagonal constraints (alias difference constraints) between clocks of the same polarity and *sum constraints* between clocks of *opposite polarity*. Additionally, we consider the extension of ECA$^+$, denoted by ECA^{++}, where the polarity requirements in the diagonal and sum constraints are relaxed. We show that ECA$^+$ are more expressive than ECA and that they can be translated in exponential time into equivalent TA. Differently from ECA$^+$, ECA^{++} are a very powerful formalism having an undecidable nonemptiness problem.

Here, we adopt a propositional-based approach where the input alphabet is given by $2^{\mathcal{P}}$ for a given set of atomic propositions. The set $C_{\mathcal{P}}$ of event clocks associated with $\mathcal{P}$ is given by $C_{\mathcal{P}} := \bigcup_{p \in \mathcal{P}} \{\overleftarrow{c}_p, \overrightarrow{c}_p\}$. Thus, for each proposition $p \in \mathcal{P}$, there are two event clocks: the *event-recording or past clock* $\overleftarrow{c}_p$ which records the time elapsed since the last occurrence of p in the input word (if any), and the *event-predicting or future clock* $\overrightarrow{c}_p$ which provides the time required to the next occurrence of p (if any). A special value $\perp$ is exploited to denote the absence of a past (resp., future) occurrence of proposition p. Formally, the values of the event clocks at a position i of a timed word w can be deterministically determined as follows.

▶ **Definition 8** (Determinisitic clock valuations). *An* event-clock valuation *is a mapping* $val : C_{\mathcal{P}} \mapsto \mathbb{R}_+ \cup \{\perp\}$*, assigning to each event clock a value in* $\mathbb{R}_+ \cup \{\perp\}$*. For a timed word* $w = (\sigma, \tau)$ *over* $2^{\mathcal{P}}$ *and a position* $0 \leq i < |w|$*, the* event-clock valuation val_i^w*, specifying the values of the event clocks at position* i *along* w*, is defined as follows for each* $p \in \mathcal{P}$*:*

$$val_i^w(\overleftarrow{c}_p) = \begin{cases} \tau_i - \tau_\ell & \textit{if there exists the unique } 0 \leq \ell < i : p \in \sigma(\ell) \textit{ and} \\ & \quad \forall k : \ell < k < i \Rightarrow p \notin \sigma(k) \\ \bot & \textit{otherwise} \end{cases}$$

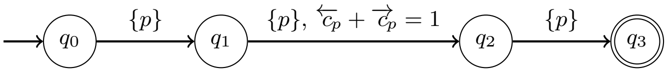
$$val_i^w(\overrightarrow{c}_p) = \begin{cases} \tau_\ell - \tau_i & \textit{if there exists the unique } i < \ell < |\sigma| : p \in \sigma(\ell) \textit{ and} \\ & \quad \forall k : i < k < \ell \Rightarrow p \notin \sigma(k) \\ \bot & \textit{otherwise} \end{cases}$$

An ECA$^+$ over $2^{\mathcal{P}}$ is a tuple $\mathcal{A} = (2^{\mathcal{P}}, Q, Q_0, C_{\mathcal{P}}, \Delta, F)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states, and Δ is a finite set of transitions (q, a, θ, q'), where $q, q' \in Q$, $a \in 2^{\mathcal{P}}$, and θ is an ECA$^+$ *event-clock constraint* that is a conjunction of atomic formulas of the following forms, where $p, p' \in \mathcal{P}$, $\sim \in \{<, \leq, \geq, >\}$, and $n_\bot \in \mathbb{N} \cup \{\bot\}$: (i) $\overleftarrow{c}_p \sim n_\bot$ or $\overrightarrow{c}_p \sim n_\bot$ (*simple constraints*); or (ii) $\overleftarrow{c}_p - \overleftarrow{c}_{p'} \sim n_\bot$ or $\overrightarrow{c}_p - \overrightarrow{c}_{p'} \sim n_\bot$ (*diagonal constraints* between event clocks of the same polarity); or (iii) $\overleftarrow{c}_p + \overrightarrow{c}_{p'} \sim n_\bot$ (*sum constraints* between event clocks of opposite polarity). We denote by $K_{\mathcal{A}}$ the maximal constant used in the event-clock constraints of $\mathcal{A}$. An ECA [3] is an ECA$^+$ which does *not* use diagonal and sum constraints. We also consider the extension of ECA$^+$, denoted by ECA^{++}, where the transition guards also exploit as conjuncts diagonal (resp., sum) constraints over event clocks of opposite polarity (resp., of the same polarity).

Let us fix an event-clock valuation *val*. We extend in the natural way the valuation *val* to differences (resp., sums) of event clocks: for all $c, c' \in C_{\mathcal{P}}$, $val(c - c') = val(c) - val(c')$ and $val(c + c') = val(c) + val(c')$ where each sum or difference involving $\bot$ evaluates to $\bot$. Given an event-clock constraint θ, *val* satisfies θ, written $val \models \theta$, if for each conjunct $t \sim n_\bot$ of θ, either (i) $val(t) \neq \bot$, $n_\bot \neq \bot$, and $val(t) \sim n_\bot$, or (ii) $val(t) = \bot$, $n_\bot = \bot$, and $\sim \in \{\leq, \geq\}$.

A run π of an ECA$^+$ (resp., ECA^{++}) $\mathcal{A}$ over a timed word $w = (\sigma, \tau)$ is a sequence of states $\pi = q_0, \ldots, q_{|w|}$ such that $q_0 \in Q_0$ and for all $0 \leq i < |w|$, $(q_i, \sigma(i), \theta, q_{i+1}) \in \Delta$ for some constraint θ such that $val_i^w \models \theta$. The run π is *accepting* if $q_{|w|} \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ of $\mathcal{A}$ is the set of timed words w over $2^{\mathcal{P}}$ s.t. there is an accepting run of $\mathcal{A}$ on w.

As an example, let us consider the ECA$^+$ $\mathcal{A}_p$, depicted below, whose set $\mathcal{P}$ of atomic propositions consists of a unique proposition p. Evidently, $\mathcal{A}_p$ accepts the set of timed words



w of length 3 of the form $(\{p\}, \tau_0), (\{p\}, \tau_1), (\{p\}, \tau_2)$ such that the time difference between the first and last symbol is 1, i.e. $\tau_2 - \tau_0 = 1$. One can easily show that there is no ECA accepting $\mathcal{L}_T(\mathcal{A}_p)$. Hence, we obtain the following result.

▶ **Theorem 9.** *For a proposition p, there is a timed language over $2^{\{p\}}$ which is definable by ECA$^+$ but is* not *definable by ECA. Hence, ECA$^+$ are strictly more expressive than ECA.*

Like ECA [3], we show that the class of timed languages accepted by ECA$^+$ (resp., ECA^{++}) is closed under Boolean operations. The closure under complementation is crucially based on the fact that event-clock values are determined solely by the input word.

▶ **Theorem 10** (Closure properties). *Given two ECA$^+$ (resp., ECA^{++}) $\mathcal{A}$ and $\mathcal{A}'$ over $2^{\mathcal{P}}$ with n and n' states, respectively, one can construct ECA$^+$ (resp., ECA^{++}) $\mathcal{A}_\cup$, $\mathcal{A}_\cap$, and $\mathcal{A}_c$ such that: (i) $\mathcal{A}_\cup$ (resp., $\mathcal{A}_\cap$) accepts $\mathcal{L}_T(\mathcal{A}) \cup \mathcal{L}_T(\mathcal{A}')$ (resp., $\mathcal{L}_T(\mathcal{A}) \cap \mathcal{L}_T(\mathcal{A}')$) and has $n + n'$ (resp., nn') states and greatest constant* $\max(K_{\mathcal{A}}, K_{\mathcal{A}'})$*; and (ii) $\mathcal{A}_c$ accepts the complement of $\mathcal{L}_T(\mathcal{A})$ and has $2^{O(n)}$ states and greatest constant $K_{\mathcal{A}}$.*

It is known that ECA can be translated in singly exponential time into equivalent TA [3]. We generalize this result to the class of ECA$^+$.

▶ **Theorem 11** (From ECA$^+$ to TA). *Given an ECA$^+$ $\mathcal{A}$ over $2^{\mathcal{P}}$, one can construct in exponential time a TA $\mathcal{A}'$ over $2^{\mathcal{P}}$ such that $\mathcal{L}_T(\mathcal{A}') = \mathcal{L}_T(\mathcal{A})$ and $K_{\mathcal{A}'} = K_{\mathcal{A}}$. Moreover, $\mathcal{A}'$ has $n \cdot 2^{O(p)}$ states and $O(p)$ clocks, where n is the number of $\mathcal{A}$-states and p is the number of event-clock atomic constraints used by $\mathcal{A}$.*

Sketched proof. Let $\mathcal{A} = (2^{\mathcal{P}}, Q, Q_0, C_{\mathcal{P}}, \Delta, F)$ be an ECA$^+$ over $2^{\mathcal{P}}$. The TA $\mathcal{A}'$ accepting $\mathcal{L}_T(\mathcal{A})$ is essentially obtained from $\mathcal{A}$ by replacing each atomic event-clock constraint of $\mathcal{A}$ with a set of standard clocks together with associated reset operations and clock constraints. To remove simple event-clock constraints of $\mathcal{A}$, we proceed as in [3]. Here, we focus on the removal of diagonal constraints over event-predicting clocks. Let us consider a diagonal predicting clock constraint $\eta : \overrightarrow{c}_p - \overrightarrow{c}_{p'} \sim n_\perp$ of $\mathcal{A}$ where $n_\perp \in \mathbb{N} \cup \{\perp\}$. We consider the case $n_\perp \neq \perp$ (the other case being simpler). For handling the constraint η, the TA $\mathcal{A}'$ exploits the fresh standard clock c_η and in case $n_\perp = 0$ and $\sim$ is $\geq$, the additional fresh standard clock $\hat{c}_\eta$. The first (resp., second) clock is reset *only if* proposition p' (resp., p) occurs in the current input symbol. Assume that the prediction η is done by $\mathcal{A}$ at position i of the input word for the first time. Then, the simulating TA $\mathcal{A}'$ carries the obligation η in its control state in order to check that there are next positions where p and p' occur and $\tau_p - \tau_{p'} \sim n_\perp$ holds, where τ_p (resp., $\tau_{p'}$) is the timestamp associated with the first next position $i_p > i$ (resp., $i_{p'} > i$) where p (resp., p') occurs. Note that all the predictions η done by $\mathcal{A}$ before positions i_p and $i_{p'}$ correspond to the same obligation. First, assume that the first next position $i_{p'} > i$ where p' occurs strictly precedes position i_p. In this case, on reading position $i_{p'}$, $\mathcal{A}'$ resets the clock c_η and replaces the old obligation η with the updated obligation (η, p') in order to check that the constraint $c_\eta \sim n_\perp$ holds when the next p occurs (i.e., at position i_p). If a new prediction η is done at a position $j_{new} \geq i_{p'}$ strictly preceding i_p, the fresh obligation η is carried in the control state together with the obligation (η, p'). We distinguish two cases:

- p' occurs in some position strictly following j_{new} and strictly preceding i_p. Let j' be the smallest of such positions. On reading position j', $\mathcal{A}'$ replaces the old obligations η and (η, p') with (η, p') and resets the clock c_η *iff* η is a lower bound constraint, i.e., $\sim \in \{>, \geq\}$. This is safe since if η is a lower bound, then the fulfillment of prediction η at j_{new} guarantees the fulfillment of prediction η at position i. Vice versa, if η is an upper bound, then the fulfillment of prediction η at i guarantees the fulfillment of prediction η' at position j_{new}. Thus, when η is a lower bound, new obligations (η, p') rewrite the old ones, while when η is an upper bound, new obligations (η, p') are ignored.
- there is no position strictly following j_{new} and strictly preceding i_p, where p' occurs. In this case, when i_p is read, the old obligation η is replaced with the obligation (η, p) unless p' occurs at position i_p (in the latter case, $\mathcal{A}'$ simply checks that $0 \sim n_\perp$).

In both the cases on reading position i_p, the constraint $c_\eta \sim n_\perp$ is checked and the obligation (η, p') is discarded. The case where $i_{p'} = i_p$ is trivial (on reading position i, $\mathcal{A}'$ checks that $0 \sim n_\perp$ holds). Finally, assume that i_p strictly precedes i'_p. The cases where either $c \neq 0$ or $\sim$ is distinct from $\geq$ are easy to handle, since in these cases if η is a lower bound (resp., upper bound), then the prediction η done at position i is not satisfied (resp., is satisfied). Thus, we focus on the case where $c = 0$ and $\sim$ is $\geq$. On reading position i_p, the clock $\hat{c}_\eta$ is reset and the old obligation η is replaced with the updated obligation (η, p) in order to check that the constraint $\hat{c}_\eta = 0$ holds when the next p' occurs (i.e., at position $i_{p'}$). In this case, new obligations (η, p) occurring before position $i_{p'}$ are ignored, i.e., the clock $\hat{c}_\eta$ is not reset at such positions. Finally, in order to ensure that raised obligations about η are eventually checked, the accepting states of $\mathcal{A}'$ do not contain such obligations. ◀

Theorem 11 cannot be extended to the class of ECA^{++}. In fact we show that for these automata, the nonemptiness problem is undecidable. The undecidability proof is similar to the one for the strong minimal TP problem.

▶ **Theorem 12.** *The nonemptiness problem of ECA^{++} is undecidable even for the subclass of ECA^{++} which use only simple atomic event-clock constraints and diagonal constraints over event clocks of opposite polarity of the form $\overleftarrow{c}_p - \overrightarrow{c}_{p'} = 0$.*

4.2 Solving the weak minimal TP problem

In this section, by exploiting the results of Section 4.1, we establish the following result, where for a TP domain $\mathcal{D} = (SV, R)$, the *maximal constant* $K_{\mathcal{D}}$ of $\mathcal{D}$ is the greatest integer occurring in the atoms of R and in the constraint functions of the variables in SV.

▶ **Theorem 13.** *Given a TP domain $\mathcal{D} = (SV, R)$, one can build in exponential time a TA $\mathcal{A}_{\mathcal{D}}$ with $2^{O(N+\sum_{x\in SV}|V_x|)}$ states, $O(N+|SV|)$ clocks, and maximal constant $O(K_{\mathcal{D}})$, where N is the overall number of quantifiers and atoms in the rules of R, such that $\mathcal{L}_T(\mathcal{A}_{\mathcal{D}}) \neq \emptyset$* iff *there is a* weak minimal plan *of $\mathcal{D}$. Moreover, the weak minimal TP problem is* **PSPACE***-complete.*

Sketched proof. For each $x \in SV$, let $x = (V_x, T_x, D_x)$. In order to prove the first part of Theorem 13, we first define an encoding of the multi-timelines of SV by means of timed words over $2^{\mathcal{P}}$ for the set $\mathcal{P}$ of propositions given by $\{\mathsf{init}\} \cup \bigcup_{x\in SV} \mathcal{P}_x$ where for each $x \in SV$, $\mathcal{P}_x = \{x\} \times V_x \times \{\mathsf{s}, \mathsf{e}\} \times \{0, 1\}$. We use the propositions in $\mathcal{P}_x$ to encode the tokens tk along a timeline for x: the start point and end point of tk are specified by propositions (x, v, s, b) and (x, v, e, b), respectively, where $b \in \{0, 1\}$ and v is the value of tk. The meaning of the bit $b \in \{0, 1\}$ is explained below. The additional proposition $\mathsf{init} \in \mathcal{P}$ is used to mark the first point of a multi-timeline code in order to check point atoms of trigger rules by ECA^{+} event-clock constraints. A *code for a timeline for x* is a timed word w over $2^{\mathcal{P}_x}$ of the form

$$w = (\{(x, v_0, \mathsf{s}, b_0)\}, \tau_0), (\{(x, v_0, \mathsf{e}, b_0)\}, \tau_1) \cdots (\{(x, v_n, \mathsf{s}, b_n)\}, \tau_n), (\{(x, v_n, \mathsf{e}, b_n)\}, \tau_{n+1})$$

such that for all $0 \le i \le n$: (i) $v_{i+1} \in T_x(v_i)$ if $i < n$; (ii) $\tau_0 = 0$ and $\tau_{i+1} - \tau_i \in D_x(v_i)$; (iii) let ℓ_i be the greatest index $0 \le j < i$ such that $v_j = v_i$ if such an index exists, and let $\ell_i := \bot$ otherwise. Then, $b_i = (b_{\ell_i} + 1) \mod 2$ if $\ell_i \neq \bot$, and $b_i = 0$ otherwise. Intuitively, for each value $v \in V_x$ occurring along w, the associated bit acts as a modulo 2 counter which is incremented at each visit of v along w (in the handling of the trigger rules under the weak minimal semantics, it is used by ECA^{+} event-clock constraints to reference the end-event of a token whose start-event (x, v, s) is the first occurrence of (x, v, s, b) for some $b \in \{0, 1\}$ after the current input position). The timed word w encodes the timeline for x of length $n + 1$ given by $\pi = (v_0, \tau_1 - \tau_0) \dots (v_n, \tau_{n+1} - \tau_n)$. Note that since the duration of a token is not zero, we have that $\tau_{i+1} > \tau_i$ for all $0 \le i \le n$. A *code for a multi-timeline for SV* is obtained by merging different timelines (one for each variable $x \in SV$), i.e., it is a non-empty timed word w over $2^{\mathcal{P}}$ of the form $w = (P_0, \tau_0) \cdots (P_n, \tau_n)$ such that: (i) for all $x \in SV$, the timed word obtained from $(P_0 \cap \mathcal{P}_x, \tau_0) \cdots (P_n \cap \mathcal{P}_x, \tau_n)$ by removing the pairs $(\emptyset, \tau_i)$ is a code of a timeline for x; (ii) $\mathsf{init} \in P_0$, $\mathsf{init} \notin P_i$ for all $1 \le i \le n$, and $P_0 \cap \mathcal{P}_x \neq \emptyset$ for all $x \in SV$.

The trigger rules in R under the weak minimal semantics can be handled by ECA^{+} over $2^{\mathcal{P}}$: the start and end points of the chosen non-trigger tokens are mapped to last and next occurrences of propositions in $\mathcal{P}$ w.r.t. the current input position (trigger) of a multi-timeline encoding, while the atoms in the rules are mapped to ECA^{+} event-clock constraints. Note that ECA^{+} cannot express trigger-less rules since the semantics of these rules does not constraint the chosen punctual events to be closest as possible to a reference event.

▷ Claim 1. One can construct in exponential time an ECA^+ $\mathcal{A}_\forall$ over $2^{\mathcal{P}}$ such that for each multi-timeline Π of SV and encoding w_Π of Π, w_Π is accepted by $\mathcal{L}_T(\mathcal{A}_\forall)$ iff Π satisfies the trigger rules in R under the weak minimal semantics. Moreover, $\mathcal{A}_\forall$ has a unique state, $O(N_a)$ atomic event-clock constraints, and maximal constant $O(K_{\mathcal{D}})$, where N_a is the overall number of atoms in the trigger rules in R.

For the trigger-less rules in R, the following result (Claim 2) has been established in [5] for a slightly different encoding of the multi-timelines. The result can be easily adapted to the encoding proposed here.

▷ Claim 2. One can construct in exponential time a TA $\mathcal{A}_\exists$ over $2^{\mathcal{P}}$ accepting the codes of the multi-timelines of SV which satisfy the *trigger-less* rules in R. Moreover, $\mathcal{A}_\exists$ has $2^{O(N_q+\sum_{x\in SV}|V_x|)}$ states, $O(|SV|+N_q)$ clocks, and maximal constant $O(K_{\mathcal{D}})$, where N_q is the overall number of quantifiers in the trigger-less rules of R.

By Theorem 11 and Claim 1–2, the first part of Theorem 13 concerning the construction of the TA $\mathcal{A}_{\mathcal{D}}$ for the TP domain $\mathcal{D}$, directly follows. For the second part of Theorem 13, we recall that non-emptiness of a TA $\mathcal{A}$ can be solved by an **NPSPACE** search algorithm in the *region graph* of $\mathcal{A}$ which uses space logarithmic in the number of states of $\mathcal{A}$ and polynomial in the number of clocks and in the length of the encoding of the maximal constant of $\mathcal{A}$ [2]. Thus, since $\mathcal{A}_{\mathcal{D}}$ can be built on the fly, and the search in the region graph of $\mathcal{A}_{\mathcal{D}}$ can be done without explicitly constructing $\mathcal{A}_{\mathcal{D}}$, membership in **PSPACE** of the weak minimal TP problem follows. **PSPACE**-hardness is proved by a polynomial time reduction from a domino-tiling problem for grids with rows of linear length [22]. ◀

5 Conclusions

We have addressed the TP problem in the dense-time setting under two novel semantics of the trigger rules: the weak and strong minimal ones. Surprisingly, we have shown that, despite the apparently small difference in the two semantics, the strong minimal one leads to an undecidable TP problem, while the weak minimal one leads to a **PSPACE**-complete TP problem. In order to solve the weak minimal TP problem, we have investigated two novel and strictly more expressive extensions of ECA which we believe to be interesting per sé in the field of TA. As for future work, we shall study the strong minimal TP problem when just one or two state variables are used, whose decidability remains an open issue. Moreover, we aim at investigating the TP problem in the controllability setting, where the values of some variables are not under the system control, but depend on the environment.

References

1 R. Alur and T. A. Henzinger. A Really Temporal Logic. *Journal of the ACM*, 41(1):181–204, 1994.

2 Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3 Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-Clock Automata: A Determinizable Class of Timed Automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999.

4 J. Barreiro, M. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, J. Ong, E. Remolina, T. Smith, and D. Smith. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proc. of the 4th ICKEPS*, 2012.

5 L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. Complexity of Timeline-Based Planning over Dense Temporal Domains: Exploring the Middle Ground. In *Proc. of the 9th GandALF 2018*, EPTCS 277, pages 191–205, 2018.

6 L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. Decidability and Complexity of Timeline-Based Planning over Dense Temporal Domains. In *Proc. of the 16th KR*, pages 627–628. AAAI Press, 2018.

7 L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. Undecidability of future timeline-based planning over dense temporal domains. *arxiv.org/abs/1904.09184*, 2019. `arXiv:1904.09184`.

8 L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and G. J. Woeginger. Timeline-Based Planning over Dense Temporal Domains with Trigger-less Rules is NP-Complete. In *Proc. of the 19th ICTCS*, volume 2243 of *CEUR Workshop Proceedings*, pages 116–127, 2018.

9 A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and N. Policella. An Innovative Product for Space Mission Planning: An A Posteriori Evaluation. In *Proc. of the 17th ICAPS*, pages 57–64, 2007.

10 A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible Timeline-Based Plan Verification. In *Proc. of the 32nd KI*, LNCS 5803, pages 49–56. Springer, 2009.

11 A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Analyzing Flexible Timeline-Based Plans. In *Proc. of the 19th ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 471–476. IOS Press, 2010.

12 S. Chien, D. Tran, G. Rabideau, S.R. Schaffer, D. Mandl, and S. Frye. Timeline-Based Space Operations Scheduling with External Constraints. In *Proc. of the 20th ICAPS*, pages 34–41. AAAI, 2010.

13 M. Cialdea Mayer and A. Orlandini. An Executable Semantics of Flexible Plans in Terms of Timed Game Automata. In *Proc. of the 22nd TIME*, pages 160–169. IEEE Computer Society, 2015.

14 M. Cialdea Mayer, A. Orlandini, and A. Ubrico. A Formal Account of Planning with Flexible Timelines. In *Proc. of the 21st TIME*, pages 37–46. IEEE Computer Society, 2014.

15 M. Cialdea Mayer, A. Orlandini, and A. Umbrico. Planning and Execution with Flexible Timelines: a Formal Account. *Acta Informatica*, 53(6–8):649–680, 2016.

16 A. Cimatti, A. Micheli, and M. Roveri. Timelines with Temporal Uncertainty. In *Proc. of the 27th AAAI*, 2013.

17 M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

18 J. Frank and A. Jónsson. Constraint-based Attribute and Interval Planning. *Constraints*, 8(4):339–364, 2003.

19 G. Geeraerts, J.F. Raskin, and N. Sznajder. Event Clock Automata: From Theory to Practice. In *Proc. of the 9th FORMATS*, LNCS 6919, pages 209–224. Springer, 2011.

20 N. Gigante, A. Montanari, M. Cialdea Mayer, and A. Orlandini. Timelines are Expressive Enough to Capture Action-based Temporal Planning. In *Proc. of the 23rd TIME*, pages 100–109. IEEE Computer Society, 2016.

21 N. Gigante, A. Montanari, M. Cialdea Mayer, and A. Orlandini. Complexity of Timeline-Based Planning. In *Proc. of the 27th ICAPS*, pages 116–124. AAAI Press, 2017.

22 D. Harel. *Algorithmics: The spirit of computing.* Wesley, 2nd edition, 1992.

23 A. K. Jónsson, P. H. Morris, N. Muscettola, K. Rajan, and B. D. Smith. Planning in Interplanetary Space: Theory and Practice. In *Proc. of the 5th AIPS*, pages 177–186. AAAI, 2000.

24 M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., 1967.

25 N. Muscettola. HSTS: Integrating Planning and Scheduling. In *Intelligent Scheduling*, pages 169–212. Morgan Kaufmann, 1994.

26 J. Ouaknine and J. Worrell. On Metric Temporal Logic and Faulty Turing Machines. In *Proc. of the 9th FOSSACS*, LNCS 3921, pages 217–230. Springer, 2006.

27 J. Ouaknine and J. Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.

28 J. Rintanen. Complexity of Concurrent Temporal Planning. In *Proc. of the 17th ICAPS*, pages 280–287. AAAI, 2007.

Dynamics on Games: Simulation-Based Techniques and Applications to Routing

Thomas Brihaye
Université de Mons, Mons, Belgium
thomas.brihaye@umons.ac.be

Gilles Geeraerts
Université libre Bruxelles, Brussels, Belgium
gigeerae@ulb.ac.be

Marion Hallet
Université de Mons, Mons, Belgium
Université libre Bruxelles, Brussels, Belgium
marion.hallet@umons.ac.be

Benjamin Monmege
Aix Marseille Univ, CNRS, LIS, Université de Toulon, France
benjamin.monmege@univ-amu.fr

Bruno Quoitin
Université de Mons, Mons, Belgium
bruno.quoitin@umons.ac.be

Abstract

We consider multi-player games played on graphs, in which the players aim at fulfilling their own (not necessarily antagonistic) objectives. In the spirit of evolutionary game theory, we suppose that the players have the right to repeatedly update their respective strategies (for instance, to improve the outcome w.r.t. the current strategy profile). This generates a dynamics in the game which may eventually stabilise to an equilibrium. The objective of the present paper is twofold. First, we aim at drawing a general framework to reason about the termination of such dynamics. In particular, we identify preorders on games (inspired from the classical notion of simulation between transitions systems, and from the notion of graph minor) which preserve termination of dynamics. Second, we show the applicability of the previously developed framework to interdomain routing problems.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases games on graphs, dynamics, simulation, network

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.35

Related Version A full version of the paper is available at [1], `http://arxiv.org/abs/1910.00094`.

Funding This article is based upon work from COST Action GAMENET CA 16228 supported by COST (European Cooperation in Science and Technology).
Gilles Geeraerts: Supported by an ARC grant of the Fédération Wallonie-Bruxelles.
Marion Hallet: Supported by an FNRS grant and an UMONS grant of the Fonds Franeau Mobilité.
Benjamin Monmege: Partially supported by the DeLTA project (ANR-16-CE40-0007) and by ANR project Ticktac (ANR-18-CE40-0015).

Acknowledgements We thank Timothy Griffin and Marco Chiesa for fruitful discussions.

1 Introduction

Games are nowadays a well-established model to reason about several problems in computer science. In the game paradigm, several agents (called *players*) are assumed to be rational, and interact in order to reach a fixed objective. As such, games have found numerous applications,

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 35; pp. 35:1–35:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

such as controller synthesis [14, 17] or network protocols [12]. In this paper, we are mainly concerned about *multi-player games played on graphs*, in which $n \geq 2$ players interact trying to fulfil their own objectives (which are not necessarily antagonistic to the others); and where the arena (defining the possible actions of the players) is given as a finite graph.

An example of such game is given in Figure 1, modelling an instance of an *interdomain routing problem* which is typical of the Internet. In this case, two service providers v_1 and v_2 want to route packets to a target node $v_\perp$ through the links that are represented by the graph edges. For economical reasons, v_1 prefers to route the traffic to $v_\perp$ through v_2 (using path c_1s_2) instead of sending them directly to $v_\perp$, and symmetrically for v_2. (Assume for instance that both v_1 and v_2 are located in Europe, and that $v_\perp$ is in America. Then, s_1 and s_2 are transatlantic links that incur a huge cost of operation for the origin nodes.) Then, assume that, initially, v_1 and v_2 route the packets through s_1 and s_2 respectively, and broadcast this information through the network. When v_1 becomes aware of the choice of v_2, he could decide to rely on the c_1 link instead, trying to route his packets through v_2. However, due to the asynchronous nature of the network, v_2 could decide to route through c_2 before the new choice of v_1 reaches it. Hence, the packets get blocked in a cycle $c_1c_2c_1\cdots$ and do not reach $v_\perp$ anymore. Then, v_1 and v_2 could decide simultaneously to reverse to s_1 and s_2 respectively which brings the network to its initial state, where the same behaviour can start again. Clearly, such oscillations in the routing policies must be avoided.

This simple example illustrates the main notions we will consider in the paper. We study the notion of *dynamics in games*, which model the behaviour of the players when they repeatedly update their strategy (i.e. their choices of actions) in order to achieve a better outcome. Then, the main objectives of the paper are to *draw a general framework to reason about the termination of such dynamics* and to *show its applicability to interdomain routing problems* (as sketched above). We say that a dynamics terminates when the players converge to an *equilibrium*, i.e. a state in which they have no incentive to further update their respective strategies. Our framework is introduced in Section 3 and 4. It relies on notions of *preorders*, in particular the *simulation* preorder [11]. Simulations are usually defined on transition systems: intuitively, a system A simulates a system B if each step of B can be mimicked in A. We consider two kinds of preorders: preorders defined on *game graphs*, i.e. on the structure of the games; and simulation defined on the *dynamics*, which are useful to reason about termination (indeed, if a dynamics D_1 simulates a dynamics D_2, and if D_1 terminates, then D_2 terminates as well). We show how the existence of a relation between *game graphs* implies the existence of a simulation between the *induced dynamics* of those games (Theorem 8, Theorem 9). This technique allows us to *check the termination of the dynamics using structural criteria about the game graph.*

The motivation of this framework comes from several examples of problems in the literature [7, 15, 9, 2] that are (sometimes implicitly) reduced to checking the termination of a dynamics in a multi-player game, and where sufficient criteria are proposed that can be expressed as the existence of a preorder between game graphs. We thus seek to unify these results, hoping that our framework will foster new applications of the game model. For instance, several sufficient conditions for termination in the network problem sketched above consist in checking that the game graph does not contain a *forbidden pattern* [7]. This containment can naturally be expressed as a preorder.

To this aim, we introduce, in Section 4 a preorder relation on game graphs, which is inspired from the classical notion of *graph minor* [10]. Intuitively, a game graph $\mathcal{G}'$ is a minor of $\mathcal{G}$ if $\mathcal{G}'$ can be obtained by deleting edges and vertices from $\mathcal{G}$ (under well-chosen conditions that are compatible with the game setting). Then, the relation "is a minor of" forms a preorder relation on game graphs and allows one to reason on the termination of dynamics (see Theorem 8 and Theorem 9).

Finally, in Section 5, we achieve our second objective, by casting questions about Interdomain Routing into our framework. Interdomain Routing is the process of constructing routes across the networks that compose the Internet. The Border Gateway Protocol (BGP), is the *de facto* standard interdomain routing protocol. As sketched in the example above, it grows a routing tree towards every destination network in a distributed manner. The example also shows that the behaviour of the BGP is naturally modelled as a game, as already pointed out before (see [5, 15] for example). In particular, checking for so-called *safety* (does the protocol always converge to a stable state?) amounts to checking termination of some dynamics. In Section 5, we formally express BGP in our game model; revisit a classical result of Sami *et al.* that we re-prove within our framework; and finally obtain a new result regarding BGP: we provide a novel necessary and sufficient condition for convergence in the restricted (yet realistic) setting where the preferences of the nodes range on the next-hop in the route only.

Due to space constraints, full proofs and some examples can be found in [1].

2 Preliminaries

Graphs. A (directed) *graph* is a pair $G = (V, E)$ where V is a set of *states* (or *nodes*), $E \subseteq V \times V$ is the set of *edges*. A *labelled graph* is a tuple $G = (V, E, L)$ where (V, E) is a graph, and $L : E \to S$ is a function associating, to each edge e, a label $L(e)$ from a set S of labels. A (labelled) graph G is finite iff V is finite. A *path* in a (labelled) graph G is a finite sequence $v_1 v_2 \cdots v_k$ or an infinite sequence $v_1 v_2 \cdots v_i \cdots$ of states such that $(v_i, v_{i+1}) \in E$ for all i. We denote v_1, the first state of a path π, by $\mathrm{first}(\pi)$. When $\pi = v_1 v_2 \cdots v_k$ is finite, we let $\mathrm{last}(\pi) = v_k$. We let $V_\perp = \{v \in V \mid \text{there is no } v' : (v, v') \in E\}$ be the set of *terminal states*. We say that a path π is *maximal* iff: either π is infinite, or π is finite and $\mathrm{last}(\pi) \in V_\perp$. Let $\pi_1 = v_1 \cdots v_k$ and $\pi_2 = u_1 u_2 \cdots$ be two paths such that $(v_k, u_1) \in E$. Then, we write $\pi_1 \pi_2$ to denote the new path $v_1 \cdots v_k u_1 u_2 \cdots$, obtained by the concatenation of π_1 and π_2.

Following automata terminologies, a labelled graph G is said to be *complete deterministic* if for every state v and label ℓ, there is exactly one edge (v, v') s.t. $L(v, v') = \ell$.

Games played on graphs. An *n-player game* is a tuple $\mathcal{G} = (V, E, (V_i)_{1 \le i \le n}, (\preceq_i)_{1 \le i \le n})$ where players are denoted by $1, \ldots, n$ and: (V, E) is a finite graph which forms the *arena* of the game, with $V_\perp$ the terminal states; $(V_i)_{1 \le i \le n}$ is a partition of $V \setminus V_\perp$ indicating which player owns each (non-terminal) state of the game (v belongs to player i iff $v \in V_i$); and $\preceq_i$ describes the preference of player i as a reflexive, transitive and total (i.e. for all π, π', $\pi \preceq_i \pi'$ or $\pi' \preceq_i \pi$) binary relation defined on maximal paths which we call *plays* (the set of all plays being denoted by Play). Intuitively, player i prefers play π to play π' iff $\pi' \preceq_i \pi$. We can extract from $\preceq_i$ a strict partial order relation by letting $\pi \prec_i \pi'$ if player i strictly prefers play π' to play π, i.e. if $\pi \preceq_i \pi'$ and $\pi' \not\preceq_i \pi$. We also write $\pi \sim_i \pi'$ if $\pi \preceq_i \pi'$ and $\pi' \preceq_i \pi$, and say that π and π' are equivalent for player i. From now on, we describe preferences by mentioning plays of interest only (implicitly, all unmentioned plays are equivalent, and below in the preference order). We also abuse notations and identify a game with its arena: so, we can write, for instance, about the "paths of $\mathcal{G}$", meaning the paths of the underlying arena.

▶ **Example 1.** Consider the example of [7]. In our context, it is modelled with the 2-player game $\mathcal{G}^{DIS} = (V, E, (V_1, V_2), (\preceq_1, \preceq_2))$ depicted on the left of Figure 1. The state $v_\perp$ is terminal. Player 1 owns $V_1 = \{v_1\}$, and player 2 owns $V_2 = \{v_2\}$. Let $E = \{c_1, s_1, c_2, s_2\}$ be such that $s_i = (v_i, v_\perp)$ and $c_1 = (v_1, v_2)$, $c_2 = (v_2, v_1)$. Edges c_i stand for "continue", and edges s_i stand for "stop". For player 1, we let the preferences be $(v_1 v_2)^\omega \prec_1 v_1 v_\perp \prec_1 v_1 v_2 v_\perp$, where π^ω denotes an infinite number of iterations of the cycle π. Symmetrically, player 2

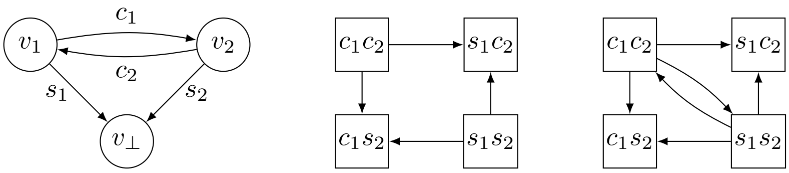


Figure 1 Left: a 2-player game $\mathcal{G}^{DIS}$. Middle: $\mathcal{G}^{DIS}\langle\xrightarrow{\mathsf{P1}}\rangle$. Right: $\mathcal{G}^{DIS}\langle\xrightarrow{\mathsf{PC}}\rangle$.

has preferences $(v_2v_1)^\omega \prec_2 v_2v_\perp \prec_2 v_2v_1v_\perp$. In this case, all unmentioned plays are equally worse for both players, in particular the plays that do not start in the state owned by the player (this will always be the case in the routing application of Section 5).

Strategies and strategy profiles. The game is played by letting players move a token along the edges of the arena. Note that, in our games, there is no designated initial state, so the play can start in any state v. The choice of the initial state is not under the control of any player. Then, the player who owns v picks an edge (v, w) and moves the token to w. It is then the turn of the player who owns w to choose an edge (w, u) and so forth. The game continues *ad infinitum* or until a terminal node has been reached, thereby forming a play. Of course, each player will act in order to yield a play that is best according to his preference order $\prec_i$. Since no player controls the choice of the initial vertex, the players will seek to obtain the best path considering *any possible initial vertex* (see the formal definitions below). This will be important for the application of Interdomain Routing in Section 5, where the games are networks and each state corresponds to a network node that wants to send a packet to one of the terminal states.

Formally, a non-maximal path is called a *history* in the following, and the set of all histories is denoted by Hist. We let Hist_i be the set of histories h such that $\mathrm{last}(h) \in V_i$, i.e. h ends in a state that belongs to player i. We further let $\mathrm{player}(h) = i$ iff $h \in \mathrm{Hist}_i$. The way players behave in the game is captured by the central notion of *strategy*, which is a mapping from a history h to a successor state in the graph, indicating how the player will play from h. A *player i strategy* is thus a function $\sigma_i \colon \mathrm{Hist}_i \to V$ such that, for all $h \in \mathrm{Hist}_i$, $(\mathrm{last}(h), \sigma_i(h)) \in E$. A *strategy profile* σ is a tuple $(\sigma_i)_{1\leq i\leq n}$ of strategies, one for each player i. In the following, when we consider a strategy profile σ, we always assume that σ_i is the corresponding strategy of player i. We also abuse notations, and write $\sigma(h)$ to denote the node obtained by playing the relevant strategy of σ from h, i.e. $\sigma(h) = \sigma_i(h)$ with $i = \mathrm{player}(h)$. We denote by $\Sigma_i(\mathcal{G})$ and $\Sigma(\mathcal{G})$ the sets of player i strategies and of strategy profiles respectively (if the game $\mathcal{G}$ is clear from the context, we may drop it and write Σ and Σ_i). As usual, given a strategy profile $\sigma = (\sigma_i)_{1\leq i\leq n}$ and a strategy σ'_j for some player j, we denote by (σ_{-j}, σ'_j) the strategy profile obtained from σ by replacing the player j strategy σ_j with σ'_j. Fixing a history h (or, in particular, an initial node) and a profile of strategies σ is sufficient to determine a unique play that is called the *outcome*: we let $\mathsf{Outcome}(\sigma, h)$ be the (unique) play $hv_1v_2\cdots$ such that for all $i \geq 1$: $v_i = \sigma(hv_1 \cdots v_{i-1})$.

Of particular interest are the *positional* strategies (sometimes called *memoryless*), i.e. the set of strategies such that the action of the player depends on the last state of the history only. That is, σ_i is positional iff for all pairs of histories h_1 and h_2 in Hist_i: $\mathrm{last}(h_1) = \mathrm{last}(h_2)$ implies $\sigma_i(h_1) = \sigma_i(h_2)$. For a *positional strategy profile* σ, and a state $v \in V$, we write $\sigma(v)$ to denote the (unique) state $\sigma(h)$ returned by σ for all h with $\mathrm{last}(h) = v$. We denote by $\Sigma^{\mathsf{P}}(\mathcal{G})$ the set of strategy profiles composed of positional strategies only, and by $\Sigma^{\mathsf{P}}_i(\mathcal{G})$ the set of player i positional strategies. From all states v, applying a positional strategy profile builds a play such that the very same decision is always taken at a particular state: therefore, it either creates a finite path without cycles, or a lasso (infinite path that starts with a finite path without cycle and continues with an infinite simple cycle, disjoint from the finite path). We let $\mathrm{Play}^{\mathsf{P}}$ be the set of all *positional plays* thus generated. In a game where we are only interested in positional strategies (as this will be the case in the application to routing, for instance), the preferences need only be defined on positional plays. Indeed, all other plays will never be obtained as an outcome, and can be assumed to be worse than any other positional play.

Game Dynamics. Let us now turn our attention to the central notion of *dynamics*. Intuitively, a dynamics consists in letting players update their strategies according to some criteria. For example, a player will want to update his strategy in order to yield a better outcome according to his preferences. Therefore, a dynamics can be understood as a graph whose states are the strategy profiles and whose edges correspond to possible updates.

▶ **Definition 2.** *Let $\mathcal{G}$ be a game. A dynamics for $\mathcal{G}$ is a binary relation $\rightarrow \subseteq \Sigma \times \Sigma$ over the strategy profiles of $\mathcal{G}$. Its associated graph is $\mathcal{G}\langle\rightarrow\rangle = (\Sigma, \rightarrow)$, where Σ is the set of states. The terminal profiles σ of $\mathcal{G}\langle\rightarrow\rangle$ (without outgoing edges) are called the* equilibria *of $\rightarrow$.*

We will focus on five dynamics, modelling certain rational behaviours of the players:

- The *one-step* dynamics $\xrightarrow{1}$. It corresponds to the minimal update that can occur, where only one player changes a single decision in order to improve the outcome from his point of view: $\sigma \xrightarrow{1} \sigma'$ iff there is a player $i \in \{1, \ldots, n\}$ and a history $h \in \mathrm{Hist}_i$ such that (i) $\sigma(h) \neq \sigma'(h)$; (ii) $\mathsf{Outcome}\,(\sigma, h) \prec_i \mathsf{Outcome}\,(\sigma', h)$; and (iii) $\sigma(h') = \sigma'(h')$ for all $h' \neq h$. Note that the equilibria of the one-step dynamics are exactly the so-called *subgame perfect equilibria* (SPE) introduced in [16] (see also [13]).
- The *positional one-step* dynamics $\xrightarrow{\mathsf{P1}}$. It ranges over positional strategy profiles only, and corresponds to a single player updating his strategy from a single state. Formally, $\sigma \xrightarrow{\mathsf{P1}} \sigma'$ (with $\sigma, \sigma' \in \Sigma^{\mathsf{P}}$) iff there are a player $i \in \{1, \ldots, n\}$ and a state $v \in V_i$ s.t. (i) $\sigma(v) \neq \sigma'(v)$; (ii) $\mathsf{Outcome}\,(\sigma, v) \prec_i \mathsf{Outcome}\,(\sigma', v)$; and (iii) $\sigma(v') = \sigma'(v')$ for all $v' \neq v$.
- The *best reply positional one-step dynamics* $\xrightarrow{\mathsf{bP1}}$. We let $\sigma \xrightarrow{\mathsf{bP1}} \sigma'$ iff there exists a player $i \in \{1, \ldots, n\}$ and a state $v \in V_i$ such that the three properties of the positional one-step dynamics are satisfied, and, in addition, the following *best-reply* condition is satisfied: (iv) for all $\sigma'' \neq \sigma'$ such that $\sigma \xrightarrow{\mathsf{P1}} \sigma''$ *if* player i is the one that has changed its strategy between σ and σ'', then: $\mathsf{Outcome}\,(\sigma'', v) \preceq_i \mathsf{Outcome}\,(\sigma', v)$.
- The *positional concurrent* dynamics $\xrightarrow{\mathsf{PC}}$ and its best reply version $\xrightarrow{\mathsf{bPC}}$. Several players can update their strategies at the same time (in a “one step” fashion), but each individual update would yield a better play when performed independently (in some sense, each player performing an update “believes” he will improve). Formally, for $\sigma, \sigma' \in \Sigma^{\mathsf{P}}$, we let $\sigma \xrightarrow{\mathsf{PC}} \sigma'$ (respectively, $\sigma \xrightarrow{\mathsf{bPC}} \sigma'$) iff for all $i \in P(\sigma, \sigma')$, $\sigma \xrightarrow{\mathsf{P1}} (\sigma'_i, \sigma_{-i})$ (respectively, $\sigma \xrightarrow{\mathsf{bP1}} (\sigma'_i, \sigma_{-i})$).

Observe that other dynamics can be defined, corresponding to other behaviours of the players. We focus on these five dynamics as they fit the applications we target in Section 5. We have already said that the equilibria of $\xrightarrow{1}$ are SPEs, and we can also see from the definitions that the equilibria of the four other dynamics coincide.

▶ **Example 3.** Let $\mathcal{G}^{\mathrm{DIS}}$ be the game from Example 1. The graphs $\mathcal{G}^{\mathrm{DIS}}\langle\xrightarrow{\mathsf{P1}}\rangle$ and $\mathcal{G}^{\mathrm{DIS}}\langle\xrightarrow{\mathsf{PC}}\rangle$ are given in the middle and the right of Figure 1, where each strategy profile is represented by the choices of the players from v_1 and v_2. For example, c_1c_2 is the strategy profile s.t.

$\sigma_1(v_1) = v_2$ and $\sigma_2(v_2) = v_1$. Note that, in this example, $\xrightarrow{\mathsf{P1}} = \xrightarrow{\mathsf{bP1}}$ and $\xrightarrow{\mathsf{PC}} = \xrightarrow{\mathsf{bPC}}$. Moreover, we can see that $\mathcal{G}^{\mathrm{DIS}}\langle\xrightarrow{\mathsf{P1}}\rangle$ has no infinite paths, contrary to $\mathcal{G}^{\mathrm{DIS}}\langle\xrightarrow{\mathsf{PC}}\rangle$. We then say that the dynamics $\xrightarrow{\mathsf{P1}}$ terminates on $\mathcal{G}^{\mathrm{DIS}}$, while $\xrightarrow{\mathsf{PC}}$ does not terminate on $\mathcal{G}^{\mathrm{DIS}}$.

The main problem we study is whether a given dynamics terminates on a certain game: we say that a dynamics $\rightarrow$ terminates on the game $\mathcal{G}$ if there is no infinite path in the graph $\mathcal{G}\langle\rightarrow\rangle$ of the dynamics. As illustrated in the introduction (Example 1), such infinite paths may be problematic in certain applications, like in the Interdomain Routing problem, where an infinite path in the dynamics means that the routing protocol does not stabilise. We are thus interested in techniques to check whether a dynamics terminates on a given game.

Sometimes, a dynamics does not terminate in general, but does when we restrict ourselves to *fair executions* where all players will eventually have the opportunity to update their strategies if they want to. Formally, given a dynamics $\rightarrow$, an infinite path $\sigma^1 \rightarrow \sigma^2 \rightarrow \cdots$ of the graph $\mathcal{G}\langle\rightarrow\rangle$ is *not fair* if there exists a player i, and a position k such that for all $\ell \geq k$, player i can switch his strategy in σ^ℓ (i.e. there is $\sigma^\ell \rightarrow \sigma'$ where $\sigma_i^\ell \neq \sigma_i'$), but for all $\ell \geq k$, player i keeps the same strategy forever (i.e. $\sigma_i^\ell = \sigma_i^k$). We say that the dynamics $\rightarrow$ *fairly terminates* for the game $\mathcal{G}$ if there are no infinite fair paths in the graph $\mathcal{G}\langle\rightarrow\rangle$: this is a weakening of the notion of termination seen before (see [1] for an example of a dynamics that does not terminate but terminates fairly).

3 Simulations: preorders on the dynamics graphs

At this point of the paper, it is important to understand that a game is characterised by two graphs: the *game graph* which gives its *structure* (see for example, Figure 1, left); and the *dynamics graph*, which, given a fixed dynamics $\rightarrow$, defines the semantics of the game as the long-term behaviour of the players (Figure 1, middle and right). In the present section, we study *preorder relations* on the *dynamics graphs*, relying on the classical notion of *simulation* [11]. They are the key ingredients to reason about the termination of dynamics.

The domain of a binary relation $R \subseteq A \times B$ is the set of elements $a \in A$ such that there exists $b \in B$ with $(a, b) \in R$. The co-domain or R is the set of elements $b \in B$ such that there exists $a \in A$ with $(a, b) \in R$. We denote the domain of R by $\mathrm{dom}(R)$. The transitive closure R^+ of relation R is defined as $(a, b) \in R^+$ iff there are $a_0 = a, a_1, a_2, \ldots, a_n = b$ such that for all $i \in \{0, 1, \ldots, n-1\}$, $(a_i, a_{i+1}) \in R$.

Partial simulations and simulations. We start with some weak version of the notion of simulation, called *partial simulation* $\sqsubseteq$. Intuitively, we say that a state u *partially simulates* a state u' (noted $u' \sqsubseteq u$) if for all successor states v' of u', the following holds: *if v' is in the domain of the simulation*, then there must be some state v simulating v' such that v is a successor of u. Formally, if $G = (V, E)$ and $G' = (V', E')$ are two graphs, a binary relation $\sqsubseteq$ contained in $V' \times V$ is a *partial simulation* of G' by G if: for all $(u', v') \in E' \cap \mathrm{dom}(\sqsubseteq)^2$, for all $u \in V$: $u' \sqsubseteq u$ *implies* there is $v \in V$ such that $(u, v) \in E$ and $v' \sqsubseteq v$. Then, a *simulation* $\sqsubseteq$ of G' by G is a partial simulation of G' by G s.t. $\mathrm{dom}(\sqsubseteq) = V'$, i.e. all states of G' are simulated by some state of G. When a (partial) simulation $\sqsubseteq$ of G' by G exists, we say that G (partially) simulates G'. The following example highlights the difference between partial simulations and simulations. Assume G with only one edge $u \rightarrow v$ and G' with only two edges $u' \rightarrow v_1'$ and $u' \rightarrow v_2'$. Then, the relation $\sqsubseteq$ s.t. $u' \sqsubseteq u$ and $v_1' \sqsubseteq v$ (but $v_2' \not\sqsubseteq v$) is a partial simulation (its domain is $\{u', v_1'\}$ so it is not a problem that v_2' is not simulated) but is not a simulation relation.

Simulations between dynamics graphs help in showing termination properties, as shown by the following folk result:

▶ **Proposition 4.** *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two games, $\rightarrow_1$ and $\rightarrow_2$ be two dynamics on $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively. If $\mathcal{G}_1\langle\rightarrow_1\rangle$ simulates $\mathcal{G}_2\langle\rightarrow_2\rangle$ and the dynamics $\rightarrow_1$ terminates on $\mathcal{G}_1$, then the dynamics $\rightarrow_2$ terminates on $\mathcal{G}_2$.*

Bisimulations and transitive closure. We can define other preorder relations on dynamics graphs. A bisimulation is a simulation $\sqsubseteq$ such that the inverse relation $\sqsubseteq^{-1}$ is also a simulation. We say that $G = (V, E)$ and $G' = (V', E')$ are bisimilar when there is a bisimulation between them. As a corollary of the previous proposition, if $\mathcal{G}_1\langle\rightarrow_1\rangle$ and $\mathcal{G}_2\langle\rightarrow_2\rangle$ are bisimilar, then $\rightarrow_1$ terminates on $\mathcal{G}_1$ if and only if $\rightarrow_2$ terminates on $\mathcal{G}_2$.

For termination purposes, it is also perfectly fine to simulate a single step of G' in several steps of G for instance. The following proposition stems from Proposition 4 and mixes the notions of transitive closures and partial simulations.

▶ **Proposition 5.** *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two games, $\rightarrow_1$ and $\rightarrow_2$ be dynamics on $\mathcal{G}_1$ and $\mathcal{G}_2$ resp.*

- *If $\mathcal{G}_1\langle\rightarrow_1^+\rangle$ simulates $\mathcal{G}_2\langle\rightarrow_2\rangle$ and the dynamics $\rightarrow_1$ terminates on $\mathcal{G}_1$, then the dynamics $\rightarrow_2$ terminates on $\mathcal{G}_2$.*
- *If $\sqsubseteq$ is a partial simulation of $\mathcal{G}_2\langle\rightarrow_2^+\rangle$ by $\mathcal{G}_1\langle\rightarrow_1^+\rangle$, and the dynamics $\rightarrow_1$ terminates on $\mathcal{G}_1$, then there are no paths in $\mathcal{G}_2\langle\rightarrow_2\rangle$ that visit a state of $\mathrm{dom}(\sqsubseteq)$ infinitely often.*

4 Minors and domination: preorders on game graphs

Let us now introduce notions of *preorders on game graphs.* We introduce a new notion of graph minor which consists in lifting the classical notion of graph minor to the context of n-player games on graphs. To the best of our knowledge, this has not been done previously. This new preorder on game graphs enables us to use in a simple context the results of Section 3 to reason about termination of dynamics. Let us start with the formal definition. For that purpose, we start by defining two transformations on game graphs. Let $\mathcal{G} = (V, E, (V_i)_i, (\preceq_i)_i)$ be an n-player game. Then we can modify it by applying either of the following transformations that yields a game $\mathcal{G}' = (V', E', (V'_i)_i, (\preceq'_i)_i)$.

- *Deletion of an edge* $(u, v) \in E$. Then, $V' = V$, $E' = E \setminus \{(u, v)\}$, $(V'_i)_i = (V_i)_i$ and $\preceq'_i$ is s.t. $\pi_1 \preceq'_i \pi_2$ iff $\pi_1 \preceq_i \pi_2$ and π_1, π_2 are both paths of $\mathcal{G}'$.
- *Deletion of a state* $v \in V_j$ (for a certain player j). This can happen in two different ways:
  1. either when v is isolated, i.e. when $(u, v) \notin E$ and $(v, u) \notin E$ for all $u \in V$. Then, $V' = V \setminus \{v\}$, $E' = E$, $V'_i = V_i$ for all $i \neq j$, $V'_j = V_j \setminus \{v\}$, and $(\preceq'_i)_i = (\preceq_i)_i$.
  2. or when v has a unique outgoing edge (v, v') and all predecessors u of v (i.e. $(u, v) \in E$) do not have v' as a successor (i.e. $(u, v') \notin E$). In this case, we have $V' = V \setminus \{v\}$, $V'_i = V_i$ for all $i \neq j$ and $V'_j = V_j \setminus \{v\}$, $E' = (E \cap (V' \times V')) \cup \{(u, v') \mid (u, v) \in E\}$, and $\pi'_1 \preceq'_i \pi'_2$ iff $\pi_1 \preceq_i \pi_2$ where π_1 and π_2 are the plays of $\mathcal{G}$ obtained from π'_1 and π'_2 respectively, by replacing all occurrences of (u, v') (for some u) by $(u, v), (v, v')$.

▶ **Definition 6.** *Let $\mathcal{G}$ and $\mathcal{G}'$ be two n-player games. Then, $\mathcal{G}'$ is a minor of $\mathcal{G}$ if $\mathcal{G}'$ can be obtained from $\mathcal{G}$ by applying a sequence of edges and states deletions.*

▶ **Example 7.** An example of minor is depicted in Figure 2. If the original preferences of the player owning state v_1 are $v_1v_4v_5v_\perp \prec v_1v_3v_\perp \prec v_1v_2v_4v_\perp \prec v_1v_2v_4v_5v_\perp$ (other plays being equally worse for this player), then after the deletion of the edge $(v_4, v_\perp)$, his preferences become $v_1v_4v_5v_\perp \prec v_1v_3v_\perp \prec v_1v_2v_4v_5v_\perp$ (the path $v_1v_2v_4v_\perp$ does not exist in the new

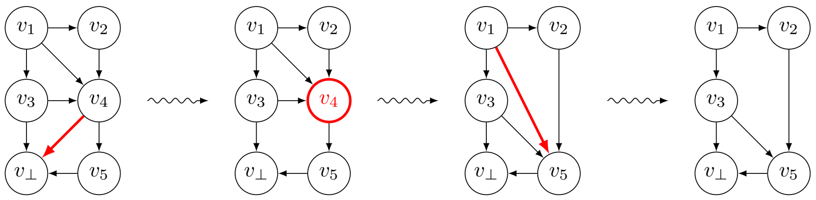


Figure 2 Minors obtained by first deleting the edge $(v_4, v_\perp)$, then the state v_4 (that has now a unique successor v_5), and then the edge (v_1, v_5).

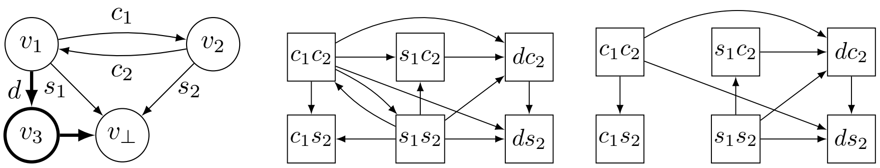


Figure 3 L: a 3-player game $\mathcal{G}$ with $\mathcal{G}^{\text{DIS}}$ (Figure 1) as a minor. M: $\mathcal{G}\langle \xrightarrow{\text{PC}} \rangle$. R: $\mathcal{G}\langle \xrightarrow{\text{bPC}} \rangle$.

graph and has simply been removed from the preferences). Next, the deletion of v_4 is allowed because it is a single outgoing edge v_5, and neither v_1 nor v_2 nor v_3 have an edge to v_5. After this deletion, the preferences become $v_1v_5v_\perp \prec v_1v_3v_\perp \prec v_1v_2v_5v_\perp$. Finally, after the deletion of the edge (v_1, v_5), the preferences become $v_1v_3v_\perp \prec v_1v_2v_5v_\perp$.

The deletion of a state therefore consists in squeezing each path of length 2 around it in a single edge. In the example, the deletion of the state v_4 consists in squeezing the paths $v_1v_4v_5$ in the edge v_1v_5, and the same for $v_2v_4v_5$ and $v_3v_4v_5$ in the edges v_2v_5 and v_3v_5 respectively. The condition $(u, v') \notin E$ makes sure that this squeezing is not perturbed by the presence of an incident edge (u, v') that could have contradictory preferences. For instance, in the previous example, we cannot remove vertex v_2 in the minor obtained before having removed edge (v_1, v_5): otherwise, we would obtain as preferences for the owner of v_1 the chain $v_1v_5v_\perp \prec v_1v_3v_\perp \prec v_1v_5v_\perp$ which is not possible.

We can link termination of dynamics on graph games to the presence of minors, in the various dynamics introduced before: if we manage to find a game minor where the dynamics does not terminate, then the original game does not terminate either.

▶ **Theorem 8.** *Let $\mathcal{G}$ be a game, and $\mathcal{G}'$ be a minor of $\mathcal{G}$. If $\to \in \{\xrightarrow{1}, \xrightarrow{\text{P1}}, \xrightarrow{\text{PC}}\}$, then $\mathcal{G}\langle\to\rangle$ simulates $\mathcal{G}'\langle\to\rangle$. In particular, via Proposition 4, if the dynamics $\to$ terminates for $\mathcal{G}$, then it terminates for $\mathcal{G}'$ too.*

Sketch of proof. We prove the result for $\xrightarrow{1}$; the two other cases are similar. Since simulations are transitive relations, it is sufficient to only consider that $\mathcal{G}'$ has been obtained from $\mathcal{G}$ either by deleting a single edge, or by deleting a single node. Let us briefly detail the case where $\mathcal{G}'$ is obtained by the deletion of a state v. If $h \in \text{Hist}(\mathcal{G}) \setminus \{h \mid \mathit{last}(h) = v\}$, we can construct a corresponding play $f(h)$ of $\mathcal{G}'$ by replacing a sequence uvv' of h by uv'. The conditions over the deletion of v implies that that f is indeed a bijection from $\text{Hist}(\mathcal{G}) \setminus \{h \mid \mathit{last}(h) = v\}$ to $\text{Hist}(\mathcal{G}')$. We then consider the following relation on strategy profiles: $\sigma' \sqsubseteq \sigma$ if for all histories $h \in \text{Hist}(\mathcal{G}) \setminus \{h \mid \mathit{last}(h) = v\}$, $\sigma'(f(h)) = \sigma(h)$ if $\sigma(h) \neq v$, and $\sigma'(f(h)) = v'$ otherwise; and show that $\sqsubseteq$ is a simulation (indeed a bisimulation). ◀

Notice that Theorem 8 suffers from three weaknesses. First, it does not hold for the best reply dynamics $\xrightarrow{\text{bP1}}$ and $\xrightarrow{\text{bPC}}$, as shown by the following example. Consider again the game $\mathcal{G}^{\text{DIS}}$ from Example 1. Further, consider the game $\mathcal{G}$ in Figure 3 obtained from $\mathcal{G}^{\text{DIS}}$ by adding a third player, who owns a single node v_3, such that the only edges to and from v_3 are (v_1, v_3) and $(v_3, v_\perp)$, and where the preferences of player 1 are now $v_1v_\perp \prec_1 v_1v_2v_\perp \prec_1 v_1v_3v_\perp$ (observe that now, he prefers a path that traverses the new node v_3 above all other paths). Clearly, $\mathcal{G}^{\mathrm{DIS}}$ is a minor of $\mathcal{G}$. Using Theorem 8, and since we know that $\mathcal{G}^{\mathrm{DIS}}\langle\xrightarrow{\mathsf{PC}}\rangle$ does not terminate, we deduce that $\mathcal{G}\langle\xrightarrow{\mathsf{PC}}\rangle$ does not terminate either. Moreover, in this example, $\mathcal{G}^{\mathrm{DIS}}\langle\xrightarrow{\mathsf{PC}}\rangle = \mathcal{G}^{\mathrm{DIS}}\langle\xrightarrow{\mathsf{bPC}}\rangle$, so even with the best-response property, the dynamics does not terminate in the minor. However, one can check that $\mathcal{G}\langle\xrightarrow{\mathsf{bPC}}\rangle$ terminates thanks to the best-response property: Player 1 will not try to obtain path $v_1v_2v_\perp$ (which leads to a cycle in $\mathcal{G}^{\mathrm{DIS}}\langle\xrightarrow{\mathsf{PC}}\rangle$), but will choose a strategy going to v_3 (see Figure 3, Right). So, $\mathcal{G}^{\mathrm{DIS}}$ is a minor of $\mathcal{G}$, s.t. $\xrightarrow{\mathsf{bPC}}$ terminates for $\mathcal{G}^{\mathrm{DIS}}$ but not in $\mathcal{G}$. The example can be adapted to $\xrightarrow{\mathsf{bP1}}$.

A second weakness is that Theorem 8 does not apply to fair termination: the dynamics $\rightarrow$ could fairly terminate for the game $\mathcal{G}$, but not for his minor $\mathcal{G}'$. This could be the case if we remove every choice (except one) for a certain player in the minor $\mathcal{G}'$ creating a fair cycle in $\mathcal{G}'$ that would not be present in $\mathcal{G}$.

Finally, the reciprocal of Theorem 8 does not hold: all dynamics terminate on the trivial graph with a single state, but it is also minor of all games, including those where the dynamics does not terminate.

This motivates the introduction of a stronger notion of graph minor, where it is allowed to remove *only* the so-called *dominated* edges. Formally, let $\mathcal{G}$ be a game, let $v \in V_i$ be a state, and let $e_1 = (v, v_1)$ and $e_2 = (v, v_2)$ be two outgoing edges of v. We say that e_1 *is dominated by* e_2 if for all *positional* strategies[1] $\sigma \in \Sigma^{\mathsf{P}}$, $\mathsf{Outcome}\,(\sigma_1, v) \prec_i \mathsf{Outcome}\,(\sigma_2, v)$, where σ_1 and σ_2 coincide with σ except that $\sigma_1(v) = v_1$ and $\sigma_2(v) = v_2$. Intuitively, this means that the player always prefers e_2 to e_1. Then, a game $\mathcal{G}'$ is said to be a *dominant minor* of $\mathcal{G}$ if it can be obtained from $\mathcal{G}$ by deleting states as before, but only deleting *dominated* edges. Equipped with this notion, we overcome the three limitations of Theorem 8 we had identified:

▶ **Theorem 9.** *Let $\mathcal{G}$ be a game and $\mathcal{G}'$ be a dominant minor of $\mathcal{G}$. If $\rightarrow \in \{\xrightarrow{\mathsf{bP1}}, \xrightarrow{\mathsf{bPC}}\}$, then we can build a simulation $\sqsubseteq$ of $\mathcal{G}'\langle\rightarrow\rangle$ by $\mathcal{G}\langle\rightarrow\rangle$ such that: (i) $\sqsubseteq^{-1}$ is a partial simulation of $\mathcal{G}\langle\rightarrow\rangle$ by $\mathcal{G}'\langle\rightarrow\rangle$; and (ii) if there is a fair cycle in $\mathcal{G}$ then there is a fair cycle in $\mathcal{G}'$.*

In particular, the dynamics $\rightarrow$ fairly terminates for $\mathcal{G}$ if and only if it does for $\mathcal{G}'$.

Now, Theorem 9 has some limitations too. We can show that it does not hold for the “non-best-reply dynamics” $\xrightarrow{\mathsf{P1}}$ and $\xrightarrow{\mathsf{PC}}$. Moreover, even when we consider best-reply dynamics, the fairness condition remains crucial: we can exhibit (see [1]) a case where there is a (non-fair) cycle in $\mathcal{G}$ but no cycles in $\mathcal{G}'$.

5 Applications to interdomain routing convergence

As explained in the introduction, the Border Gateway Protocol (BGP) is the *de facto* standard interdomain routing protocol. Its role is to establish routes to all the networks that compose the Internet. BGP does this by growing a routing tree towards every destination network in a distributed manner, as follows. In the initial state, only the router in the destination network has a route towards itself that it advertises to its neighbours. Each time a router receives an advertisement, it selects among the neighbour routes the one it considers best and then advertises it to its neighbours. The process repeats until no router wants to change

[1] We restrict our definition to the context of positional strategies, for the sake of brevity, but it can be extended to the more general setting.

its best route. To select its best route, a router first filters the received routes to retain only *permitted one*s and ranks them according to its *preference.* Both the filtering and ranking of routes by a router are decided based on the network's routing policy. For example, a route can be preferred over another because it offers better performance or costs less and it can be filtered out because it is not economically viable.

As shown in the introductory example, the routing approach at the heart of BGP has known convergence issues. It could fail to reach an equilibrium, entering a persistent oscillatory behaviour or it could have no equilibrium at all. This is a well-studied problem that has led to considerable work [7, 6, 5, 15, 3, 4, 8, 12]. In their seminal work [7], Griffin *et al.* analysed the BGP convergence properties using a simplified model named the Stable Path Problem (SPP). The main questions they ask are the following: (1) whether an SPP instance is **Solvable**, i.e., whether it admits a stable state; (2) whether the stable state is **Unique**; and (3) whether the system is **Safe**, i.e. it always converges to a stable state.

They also give a *sufficient* condition for an SPP instance to be safe: the absence of a substructure named a **Dispute Wheel**. Later, Sami *et al.* [15] have shown that the existence of multiple stable states is a sufficient condition to prevent safety (i.e. Safe $\implies$ Unique). These results have later been refined by Cittadini *et al.* [3]. While the works just cited focus on the definition of sufficient conditions for safety, another approach by Gao and Rexford [6] achieves convergence by enforcing only local conditions on route preferences.

In this section, we show how SPP can be expressed in our n-player game model, therefore **Safety** reduces to checking for termination of the game dynamics. We revisit the result of Sami *et al.* by providing a new proof that relies on our framework. Then, we further exploit this framework to obtain a new result about SPP: we provide a *necessary and sufficient* condition for safety in a setting which is more restricted (yet still realistic) than in [7].

One target games. We first translate the SPP, as a combination of: (1) a reachability game that models the network topology and routing policies; and (2) the best-reply positional concurrent dynamics that models the asynchronous behaviour of the routing protocol.

Using this approach, the routing safety problem translates to a dynamics termination problem.

We rely on a particular class of games, that we call *one target games* (1TG for short): they have a unique target, the destination network, that all players want to reach. Each player corresponds to a network in the Internet and as such owns a single state. The routing policies of networks are modelled by the preference relations and by the distinction between permitted and forbidden paths. The preferences are only over positional strategies (paths), meaning that each network picks its next-hop independently of its predecessors. Permitted and forbidden paths model the fact that only some paths are allowed by the networks routing policies. Forbidden paths are also used to take into account additional restrictions that cannot be directly modelled. In SPP, the paths are simple (no loops); and non-simple paths are forbidden, for obvious reasons of efficiency. Moreover, in SPP, if at some point a network reaches a forbidden path, he will inform his neighbours that he is not able to reach the target. To model this, we impose a requirement that that if a path is permitted, all its suffixes are also permitted.

Formally, let $\mathcal{G} = (V, E, (V_i)_{1\leq i\leq n}, (\preceq_i)_{1\leq i\leq n})$ be an n-player game. For all $1 \leq i \leq n$, we assume that $\mathcal{P}_i$ is the set of *permitted paths* of player i. All these paths are finite paths of the form $v_i \cdots v_\perp \in \mathcal{P}_i$. We denote by $\mathcal{P}_i^c$ the set of *forbidden paths*, i.e. all the positional plays starting in v_i that are not in $\mathcal{P}_i$ (in particular, all infinite paths are forbidden). We let $\mathcal{P} = \bigcup_{1\leq i\leq n} \mathcal{P}_i$ and $\mathcal{P}^c = \bigcup_{1\leq i\leq n} \mathcal{P}_i^c$. Then, $\mathcal{G}$ is a *one target game* (1TG) if:

- $V_\perp = \{v_\perp\}$, and, for all players i: $V_i = \{v_i\}$;
- for all $\pi_1 \in \mathcal{P}_i^c$, for all $\pi_2 \in \mathcal{P}_i$: $\pi_1 \prec_i \pi_2$ (permitted is better than forbidden);
- for all $\pi_1, \pi_2 \in \mathcal{P}_i^c$: $\pi_1 \sim_i \pi_2$ (all forbidden paths are equivalent);
- for all $\pi_1, \pi_2 \in \mathcal{P}_i$: $\pi_1 \sim_i \pi_2$ *implies* that then there are $v \in V$ and $\widetilde{\pi}_1, \widetilde{\pi}_2$ s.t. $\pi_1 = v_i v \widetilde{\pi}_1$ and $\pi_2 = v_i v \widetilde{\pi}_2$ (if two permitted paths are equivalent, they have the same next-hop);
- for all $\pi \in \mathcal{P}_i$, for all suffixes $\widetilde{\pi}$ of π: $\widetilde{\pi} \in \mathcal{P}$ (all suffixes of permitted paths are permitted).

Our running example (Figure 1) is a 1TG. Since, in such a game, each player owns one and only one state, we will abuse notation by confusing each state $v \in V$ with its player. For example, for $v \in V_i$, we will write $\prec_v$ instead of $\prec_i$.

Sami et al: Termination implies a unique terminal node. Equipped with this definition, we start by revisiting a result of Sami *et al.* saying that when an instance of SPP is *safe*, the solution is unique. In our setting, this translates as follows:

▶ **Theorem 10.** *Let $\mathcal{G}$ be a 1TG. If $\xrightarrow{\mathsf{bPC}}$ fairly terminates for $\mathcal{G}$ (i.e. the corresponding instance of SPP is safe), then it has exactly one equilibrium.*

We (re-)prove this result in our setting. We rely on the notion of L-fair path that we define now. For a labelled graph $G = (V, E, L)$, we write $v_1 \rightarrow_a v_2$ iff $L(v_1, v_2) = a$ (for $v_1, v_2 \in V$). We further write $v_1 \rightarrow_A v_2$ with $A = a_1 \cdots a_n$ iff $v_1 \rightarrow_{a_1} \cdots \rightarrow_{a_n} v_2$. Then, a path $\pi = v_1 v_2 \cdots$ is L-fair if all labels occur infinitely often in this path, i.e. for all $a \in L$, for all $k \geq 1$, $\exists k' \geq k$ such that $L(v_{k'}, v_{k'+1}) = a$. A cycle π is called *constant* if there exists a state v such that $\pi = v^\omega$. Moreover, a node is a *sink* if its only outgoing edges are self loops. Then, we can show the following technical lemma:

▶ **Lemma 11.** *Let $G = (V, E, L)$ be a finite complete deterministic labelled graph satisfying: for all $v \in V$, for all $a, b \in L$, there are $A, B \in L^*$ and $\tilde{v} \in V$ such that $v \rightarrow_{aA} \tilde{v}$ and $v \rightarrow_{baB} \tilde{v}$. If there exists a state from which we can reach two different sinks, then G has a non constant L-fair cycle.*

Thanks to this result, we can establish Theorem 10. We prove the contrapositive, as follows. We assume that $\mathcal{G}\langle\xrightarrow{\mathsf{bPC}}\rangle$ has more than one equilibrium. We introduce a new dynamics $\rightsquigarrow$ (taking into account the *beliefs* of the players about the other players' strategies) and we use Lemma 11, to show that $\mathcal{G}\langle\rightsquigarrow\rangle$ has an L-fair cycle. Then, we define a partial simulation $\sqsubseteq$ of $\mathcal{G}\langle\rightsquigarrow\rangle$ by $\mathcal{G}\langle\xrightarrow{\mathsf{bPC}}\rangle$ and use Proposition 5 to conclude that $\mathcal{G}\langle\xrightarrow{\mathsf{bPC}}\rangle$ has a cycle, which is fair. Hence, $\xrightarrow{\mathsf{bPC}}$ does not fairly terminate.

Griffin et al: Dispute wheels. Another classical notion in the BGP literature is that of a *dispute wheel* (DW for short), defined by Griffin *et al.* [7] as a "*circular set of conflicting rankings between nodes*". They have shown that the absence of a DW is a sufficient condition for safety, which is of course of major practical interest to prove that BGP will converge in a given network. Moreover, a DW is an instance of a *forbidden pattern* in a game, and we will thus apply the results from Section 4.

We start by formally defining a DW. Let $\mathcal{G} = (V, E, (V_i)_{1 \leq i \leq n}, (\preceq_i)_{1 \leq i \leq n})$ be a 1TG with $\mathcal{P}_i$ the set of permitted paths of v_i. A triple $D = (U, P, H)$ is a DW of $\mathcal{G}$ if:
(i) $U = (u_1, \ldots, u_k) \in V^k$ is a tuple of states; (ii) $P = (\pi_1, \ldots, \pi_k)$ is a tuple of permitted paths such that for all $1 \leq i \leq k$: $\pi_i \in \mathcal{P}_{u_i}$, i.e., π_i is a permitted path starting in u_i ; (iii) $H = (h_1, \ldots, h_k)$ is a tuple of non-maximal paths such that for all $1 \leq i \leq k$: $h_i \pi_{(i \mod k)+1} \in \mathcal{P}_{u_i}$; and (iv) for all $1 \leq i \leq k$: $\pi_i \prec_{u_i} h_i \pi_{(i \mod k)+1}$.

Intuitively, in a DW, all players u_i (for $i = 1, \dots, k$) can chose between two paths to $v_\perp$: either a "direct" path π_i, or an "indirect" path $h_i\pi_{(i \mod k)+1}$, which traverses $u_{(i \mod k)+1}$; and where the latter is always preferred. So u_1 prefers to reach through u_2, u_2 through u_3, and so on until u_k who prefers to reach through u_1. Such a conflict clearly yields loops where the target is never reached. The game in Figure 1 is a typical example of game that has a DW, if we let $U = (v_1, v_2)$, $P = (v_1v_\perp, v_2v_\perp)$ and $H = (v_1, v_2)$. Indeed, $v_1v_\perp \prec_1 v_1v_2v_\perp$ and $v_2v_\perp \prec_2 v_2v_1v_\perp$. Then, in our setting the sufficient condition of Griffin *et al.* [7] becomes:

▶ **Theorem 12** ([7]). *Let $\mathcal{G}$ be a* 1TG. *If $\mathcal{G}$ has no DW then $\xrightarrow{\mathsf{bPC}}$ fairly terminates for $\mathcal{G}$.*

New result: strong dispute wheels for a necessary condition. It is well-known, however, that the absence of a DW is *not necessary* (see for example Figure 3 for a game that has a DW but where $\xrightarrow{\mathsf{bPC}}$ terminates). As far as we know, finding a *unique and necessary* condition for the *fair* termination of $\xrightarrow{\mathsf{bPC}}$ in 1TGs is still an open problem.

Relying on our framework, we manage to obtain such a *necessary and sufficient* condition in a restricted setting. We first strengthen the definition of DW by introducing the notion of *strong dispute wheel* (SDW for short). We then obtain two original (as far as we know) results regarding SDW. First, the absence of SDW is a *necessary* condition for the termination of $\xrightarrow{\mathsf{PC}}$ (i.e. we drop the best-reply and the fairness hypothesis). Second, the absence of an SDW is also a *sufficient* condition in the restricted setting where the preferences of the players range only on their next-hop. This means for example that u_1 prefers to reach the target through u_2 rather than through u_3, but does not mind the route u_2 uses (as long as $v_\perp$ is reached). While this is a restriction, we believe that it is still meaningful in practice, since networks usually have little control about the routes chosen by their neighbours.

We first define the notion of SDW. Let $\mathcal{G}$ be a 1TG and $D = (U, P, H)$ be a DW of $\mathcal{G}$. Then, D is a *strong* dispute wheel (SDW) of $\mathcal{G}$ if:

1. for all $1 \le i \le k$: all states $u_i \in U$ occur *only* in π_i, h_i and h_{i-1} (we identify h_0 with h_k) and not in the other paths of P and H; and
2. for all $\pi_i, \pi_j \in P$, for all $h_k, h_\ell \in H$ with $k \neq \ell$: π_i, h_k and h_ℓ share no states of $V \setminus U$, and if π_i and π_j share a state v of $V \setminus U$ then π_i and π_j have the same suffix after v.

An important property of this definition is that, whenever a game $\mathcal{G}$ contains an SDW $D = (U, P, H)$, we can extract a minor $\mathcal{G}'$ which is essentially an SDW restricted to the states of U (formally, $\mathcal{G}'$ contains an SDW $D' = (U', P', H')$ where $U' = U$ is the set of states of $\mathcal{G}'$). We do so by first deleting from $\mathcal{G}$ all edges that do not occur in P and H; then all $v \notin U$ (which have at most one outgoing edge at this point), using the procedure described in Section 4. Note that the two extra conditions in the definition of an SDW guarantee that the deletion of all the states $v \notin U$ can occur.

▶ **Theorem 13.** *Let $\mathcal{G}$ be a* 1TG. *If $\xrightarrow{\mathsf{PC}}$ terminates for $\mathcal{G}$, then $\mathcal{G}$ has no SDW.*

Proof. By Theorem 8, it is sufficient to prove that the dynamics $\xrightarrow{\mathsf{PC}}$ does not terminate in the minor game $\mathcal{G}'$ extracted from the SDW (see above). We let, for all $1 \le i \le k$, $\sigma_1(u_i) = u_{(i \mod k)+1}$, and $\sigma_2(u_i) = v_\perp$. Since the path resulting from σ_1 does not visit $v_\perp$, by definition of an SDW, we have $\sigma_1 \xrightarrow{\mathsf{PC}} \sigma_2 \xrightarrow{\mathsf{PC}} \sigma_1$. Hence $\mathcal{G}'\langle\xrightarrow{\mathsf{PC}}\rangle$ contains a cycle. ◀

Thus, the absence of an SDW is a *necessary* condition for the termination of $\xrightarrow{\mathsf{PC}}$. We can further show that this condition is *sufficient* in the restricted case where any two (permitted) paths that have the same next-hop are equivalent. Formally, let $\mathcal{G}$ be a 1TG. We say that it

$$\begin{tikzcd}
\xrightarrow{\mathsf{bPC}} \text{does not fairly terminate for } \mathcal{G} \arrow[r] \arrow[d, "\text{Thm 12 [7]}"] & \xrightarrow{\mathsf{PC}} \text{does not fairly terminate for } \mathcal{G} \arrow[r] \arrow[d, "\text{[7]}"'] \arrow[dr, dashed, leftrightarrow, "\text{Thm 14}"] & \xrightarrow{\mathsf{PC}} \text{does not terminate for } \mathcal{G} \\
\mathcal{G} \text{ has a DW} & \mathcal{G} \text{ has a DW+} \arrow[l] & \mathcal{G} \text{ has an SDW} \arrow[l] \arrow[u, "\text{Thm 13}"']
\end{tikzcd}$$

Figure 4 Relationship between SDW and prior results: dashed arrow only holds for N1TG.

is a *neighbour one target game* (N1TG for short) if for all players i, for all permitted paths $\pi_1, \pi_2 \in \mathcal{P}_i$ of player i: $\pi_1 = v_i v \pi_1'$ and $\pi_2 = v_i v \pi_2'$ implies that $\pi_1 \sim_i \pi_2$. Then, we can show the following, relying on Theorem 13 (and thus, also on Theorem 8):

▶ **Theorem 14.** *Let $\mathcal{G}$ be a* N1TG*. Then, $\xrightarrow{\mathsf{PC}}$ does not fairly terminate for $\mathcal{G}$ if and only if $\mathcal{G}$ has an SDW.*

Sketch of proof. In [7], Griffin *et al.* prove a stronger result than Theorem 12, showing that *if* $\mathcal{G}\langle\xrightarrow{\mathsf{PC}}\rangle$ has a fair cycle, *then* $\mathcal{G}$ has a DW satisfying the following additional properties: (1) for all $u_i \in U$: $j \neq i$ implies $u_i \notin \pi_j$; (2) for all $v \notin U$, for all i, j: $v \notin \pi_i \cap h_j$; and (3) for all $v \in \pi_i \cap \pi_j$: $\pi_i(v) = \pi_j(v)$.

We call DW+ such DW. Then, the general schema of our proof is summarised in Figure 4: first, we show that the existence of a DW+ implies an SDW by showing the required additional properties. By Theorem 13, this implies $\mathcal{G}\langle\xrightarrow{\mathsf{PC}}\rangle$ has a cycle. Then, we conclude by showing that this implies the existence of a fair cycle. ◀

Finding an SDW in practice. Because of the intricate definition of SDW, finding an SDW in a real network may be challenging in practice. However, we have:

▶ **Proposition 15.** *Let $\mathcal{G}$ be an* N1TG*. Then $\xrightarrow{\mathsf{PC}}$ does not fairly terminate for $\mathcal{G}$ if and only if $\mathcal{G}^{\mathrm{DIS}}$ is a minor of $\mathcal{G}$.*

6 Perspectives

We envision multiple directions of future work. First, we could consider games with imperfect information. In the application to interdomain routing for example, this could be used to model a malicious router that advertises lies to selected neighbours. Advertising a non-existent or non-feasible path would allow for example an attacker to attract the packets of an opponent's network. Second, we could investigate a better way to model asynchronicity (useful for the routing problem) than the concurrent dynamics we have studied here. Third, we chose to model fairness via a qualitative property which ensures that *all the players will eventually have the opportunity to update their strategies if they want to.* An alternative way could be the use of probabilities: indeed, there are games for which a dynamics → does not fairly terminate, but where an equilibrium is reached almost surely when interpreting $\mathcal{G}\langle\rightarrow\rangle$ as a finite Markov chain (with uniform distributions). Finally, we could apply the dynamics of graph-based games to other problems than interdomain routing, like *load sensitive routing*.

References

1 T. Brihaye, G Geeraerts, M. Hallet, Benjamin Monmege, and B. Quoitin. Dynamics on Games: Simulation-Based Techniques and Applications to Routing. *CoRR*, abs/1910.00094, 2019. arXiv:1910.00094.

2 Thomas Brihaye, Gilles Geeraerts, Marion Hallet, and Stéphane Le Roux. Dynamics and Coalitions in Sequential Games. In Patricia Bouyer, Andrea Orlandini, and Pierluigi San Pietro, editors, *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September 2017.*, volume 256 of *EPTCS*, pages 136–150, 2017. doi:10.4204/EPTCS.256.10.

3 Luca Cittadini, Giuseppe Di Battista, Massimo Rimondini, and Stefano Vissicchio. Wheel+Ring=Reel: the Impact of Route Filtering on the Stability of Policy Routing. *IEEE/ACM Transaction on Networking*, 19(4):1085–1096, August 2011.

4 Matthew L. Daggitt, Alexander J. T. Gurney, and Timothy G. Griffin. Asynchronous convergence of policy-rich distributed Bellman-Ford routing protocols. In Sergey Gorinsky and János Tapolcai, editors, *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*, pages 103–116. ACM, 2018. doi:10.1145/3230543.3230561.

5 Alex Fabrikant and Christos H. Papadimitriou. The complexity of game dynamics: BGP oscillations, sink equilibria, and beyond. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 844–853. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347175.

6 Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. *IEEE/ACM Trans. Netw.*, 9(6):681–692, 2001. doi:10.1109/90.974523.

7 Timothy Griffin, F. Bruce Shepherd, and Gordon T. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Netw.*, 10(2):232–243, 2002. URL: http://portal.acm.org/citation.cfm?id=508332.

8 Aaron D. Jaggard, Neil Lutz, Michael Schapira, and Rebecca N. Wright. Dynamics at the Boundary of Game Theory and Distributed Computing. *ACM Trans. Economics and Comput.*, 5(3):15:1–15:20, 2017. doi:10.1145/3107182.

9 S. Le Roux and A. Pauly. A Semi-Potential for Finite and Infinite Sequential Games (Extended Abstract). In Domenico Cantone and Giorgio Delzanno, editors, Proceedings of the Seventh International Symposium on *Games, Automata, Logics and Formal Verification,* Catania, Italy, 14-16 September 2016, volume 226 of *Electronic Proceedings in Theoretical Computer Science*, pages 242–256. Open Publishing Association, 2016. doi:10.4204/EPTCS.226.17.

10 László Lovász. Graph minor theory. *Bull. Amer. Math. Soc. (N.S.)*, 43(1):75–86, 2006.

11 Robin Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.

12 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA, 2007.

13 Martin J. Osborne and Ariel Rubinstein. *A course in game theory.* MIT Press, Cambridge, MA, 1994.

14 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *POPL*, pages 179–190. ACM Press, 1989.

15 Rahul Sami, Michael Schapira, and Aviv Zohar. Searching for Stability in Interdomain Routing. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 549–557. IEEE, 2009. doi:10.1109/INFCOM.2009.5061961.

16 Reinhard Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfrägentragheit. *Zeitschrift für die gesamte Staatswissenschaft*, 12:201–324, 1965.

17 Wolfgang Thomas. On the Synthesis of Strategies in Infinite Games. In *STACS*, pages 1–13, 1995.

Query Preserving Watermarking Schemes for Locally Treelike Databases

Agnishom Chattopadhyay
Chennai Mathematical Institute, Chennai, India
UMI ReLaX, Indo-French joint research unit

M. Praveen
Chennai Mathematical Institute, Chennai, India
UMI ReLaX, Indo-French joint research unit

Abstract

Watermarking is a way of embedding information in digital documents. Much research has been done on techniques for watermarking relational databases and XML documents, where the process of embedding information shouldn't distort query outputs too much. Recently, techniques have been proposed to watermark some classes of relational structures preserving first-order and monadic second order queries. For relational structures whose Gaifman graphs have bounded degree, watermarking can be done preserving first-order queries.

We extend this line of work and study watermarking schemes for other classes of structures. We prove that for relational structures whose Gaifman graphs belong to a class of graphs that have locally bounded tree-width and is closed under minors, watermarking schemes exist that preserve first-order queries. We use previously known properties of logical formulas and graphs, and build on them with some technical work to make them work in our context. This constitutes a part of the first steps to understand the extent to which techniques from algorithm design and computational learning theory can be adapted for watermarking.

2012 ACM Subject Classification Security and privacy → Information accountability and usage control; Theory of computation → Finite Model Theory; Information systems → Relational database model

Keywords and phrases Locally bounded tree-width, closure under minors, first-order queries, watermarking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.36

Related Version A full version of the paper is available at `https://arxiv.org/abs/1909.11369`.

Funding *M. Praveen*: Partially supported by a grant from the Infosys foundation.

Acknowledgements The authors thank B. Srivathsan and K. Narayan Kumar for feedback on the draft.

1 Introduction

Watermarking of digital content can be used to check intellectual property violations. The idea is to embed some information, such as a binary string, in the digital content in such a way that it is not easily apparent to the end user. If the legitimate owner of the digital content suspects a copy to be stolen, they should be able to retrieve the embedded information, even with limited access to the stolen copy, even if it has been tampered to remove the embedded information. Here there are two opposing goals. One is to be able to embed large amount of information. The other is to ensure that the embedding doesn't distort the content too much.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 36; pp. 36:1–36:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 `EmployeeTable` of Ex. 1.

(a) The original `EmployeeTable`.

FirstName	City	Salary
John	Chennai	10,000
Arjun	Coimbatore	20,000
Pooja	Chennai	15,000
Neha	Vellore	30,000
Padma	Coimbatore	20,000

(b) A distorted `EmployeeTable`.

FirstName	City	Salary
John	Chennai	10,001
Arjun	Coimbatore	19,999
Pooja	Chennai	14,999
Neha	Vellore	30,000
Padma	Coimbatore	20,001

There can be many ways to measure how much distortion is acceptable. In [1], embedding is performed by flipping bits in numerical attributes while preserving the mean and variance of all numerical attributes. There are other works that focus on the specific use of the digital content: in [14], the digital content consists of graphs whose vertices represent locations and weighted edges represent distance between locations. It is shown that information can be embedded in such a way that the shortest distance between any two locations is not distorted too much.

We study embedding information in relational databases such that the distortion on query outputs is bounded.

▶ **Example 1.** The table `EmployeeTable` shown in Table 1(a) is an example of a database instance of an organization's record of employees.

Consider the following query parameterized by the variable x.

$\psi(x) \equiv$ `select FirstName,Salary from EmployeeTable where City=`x

If we substitute the variable x with a particular city c, the above query lists the salaries of individuals working in that city. Let `total`(c) be the sum of all salaries listed by the query $\psi(c)$. We want to hide data in `EmployeeTable` by distorting the `Salary` field. Let `total`$'(c)$ be the sum of all salaries listed by the query $\psi(c)$ run on the distorted database. We say that the distortion *preserves the query* $\psi(x)$ if there is a constant B such that for any city c, the absolute value of the difference between `total`(c) and `total`$'(c)$ is bounded by B.

Assuming that we can distort each employee's salary by at most 1 unit and we wish to maintain the bound B to be 0, our options are the following: increase the salary of `John` by 1 and decrease the salary of `Pooja` by 1 or vice-versa. Similarly for `Arjun` and `Padma`. This gives us four different ways distort the data base. These distortions are designed to preserve only the query $\psi(x)$. If a different query is run on the same distorted databases, the results may vary widely. Suppose the organization distributes the four distorted databases among it's branches. If a stolen copy of the database is found, the organization can run the query $\psi(x)$ on the stolen copy. By observing the salaries of `John`, `Pooja`, `Arjun` and `Padma` and comparing them with the values from the original database, one can narrow down on the branches where the leakage happened. The organization only needs to run the query $\psi(x)$ on the suspected stolen copy, just like any normal consumer of the database. This is important since the entity possessing the stolen copy may not allow full access to its copy of the database. We say a watermarking scheme is *scalable* if for larger databases, there are larger number of ways to distort the database, while still preserving queries of interest.

In [13], meta theorems are proved regarding the existence of watermarking schemes for classes of databases preserving queries written in classes of query languages. The Gaifman graph of a database is a graph whose set of vertices is the set of elements in the universe of the database. There is an edge between two elements if the two elements participate in some tuple in the database. For databases with unrestricted structures, even simple queries can't be preserved; see [13, Theorem 3.6] and [12, Example 3]. Preserving queries written in powerful query languages and handling databases with minimum restrictions on their structure are conflicting goals. For databases whose Gaifman graphs have bounded degree, first-order queries can be preserved [13]. It is also shown that for databases whose Gaifman graphs are similar to trees, MSO queries can be preserved. The similarity of a graph to a tree is measured by tree-width. For example, XML documents are trees and have tree-width 1.

Contributions. We prove that watermarking schemes exist for databases whose Gaifman graphs belong to a class of graphs that have locally bounded tree-width and is closed under minors, preserving unary first-order queries. Classes of graphs with bounded degree are contained in this class. A graph G has locally bounded tree-width if it satisfies the following property: there exists a function f such that for any vertex v and any number r, the sphere of radius r around v induces a subgraph on G whose tree-width is at most $f(r)$.

Why first-order logic? The pivotal Codd's theorem [3] states that first-order logic is expressively equivalent to relational algebra, and relational algebra is the basis of standard relational database query languages.

Why locally bounded treewidth? Classes of graphs with locally bounded treewidth are good starting points to start using techniques from algorithm design and computational learning theory in other areas. Seese [18] proved that first-order properties can be decided in linear time for graphs of bounded degree. Baker [2] showed efficient approximation algorithms for some specific hard problems, when restricted to planar graphs. Eppstein [7] showed that Baker's technique continues to work in a bigger class of graphs: it suffices for the class of graphs to have locally bounded tree-width and additionally, the class should be closed under minors. Frick and Grohe [9] showed that on any class of graphs with locally bounded tree-width, any problem definable in first-order logic can be decided efficiently[1]. For problems definable in first-order logic, the classes of graphs for which efficient algorithms exist was then extended to bigger and bigger classes: excluded minors [8], locally excluded minors [5], bounded expansion, locally bounded expansion [6] and nowhere dense [11]. It is now known that nowhere dense graphs are the biggest class of graphs for which there are efficient algorithms for first-order definable problems [6, 15, 11], provided some complexity theoretic assumption are true. Results related to computational learning theory have been proved in [12] for classes of graphs with locally bounded treewidth. Recently, similar results have been proven for nowhere dense classes of graphs [17].

Why unary queries? Some of the techniques we have used are difficult to extend to non-unary queries. Some technical details about this are discussed in the conclusion.

Related Works. The fundamental definitions of what it means for a watermarking scheme to be scalable and preserving a query was given in [14]. It was shown in [14] that on weighted graphs, scalable watermarking schemes exist preserving shortest distance between vertices. The adversarial model was also introduced in [14], where the person possessing the stolen

[1] Here, efficiency means fixed parameter tractability; see [9] for details.

copy can introduce additional distortion to evade detection. It is shown in [14] that a watermarking scheme for the non-adversarial model can be transformed to work for the adversarial model, under some assumption about the quantity of distortion introduced by the person trying to evade detection, and the amount of knowledge the person possesses. Gross-Amblard [13] adapted these definitions for relational structures of any vocabulary and any query written in Monadic Second Order (MSO) logic, and showed results about classes of structures of bounded degree and tree-width. Gross-Amblard [13] also provided the insight that existence of scalable watermarking schemes preserving queries from a certain language is closely related to learnability of queries in the same language. We make use of this insight in our work. Grohe and Turán [12] proved that MSO-definable families of sets in graphs of bounded tree-width have bounded Vapnik-Chervonenkis (VC) dimension, which has well known connections in computational learnability theory. It is also shown in [12] that on classes of graphs with locally bounded tree-width, first-order definable families of sets have bounded VC dimension.

2 Preliminaries

Relational databases. A signature (or database schema) τ is a finite set of relation symbols $\{R_1, \dots, R_t\}$. We denote by r_i the arity of R_i for every $i \in \{1, \dots, t\}$. A τ-structure $G = (V, R_1^G, \dots R_t^G)$ (or database instance) consists of a set V called the universe, and an interpretation $R_i^G \subseteq V^{r_i}$ for every relation symbol R_i. For a fixed $s \in \mathbb{N}$, a *weighted structure* (G, W) is a finite structure G together with a weight function W, which is a partial function from V^s to $\mathbb{N}$, that maps a s-tuple $\overline{b}$ to its weight $W(\overline{b})$.

First Order and Monadic Second Order Queries. An atomic formula is a formula of the form $x = y$ or $R(x_1, \dots x_r)$, where $x, y, x_1 \dots x_r$ are variables and R is an r-ary relational symbol in τ. First-order (FO) formulas are formulas built from atomic formulas using the usual boolean connectives and existential and universal quantification over the elements of the universe of a structure.

Monadic Second Order (MSO) logic extends first-order logic by allowing existential and universal quantifications over subsets of the universe. Formally, there are two types of variables. Individual variables, which are interpreted by elements of the universe of a structure, and set variables, which are interpreted by subsets of the universe of a structure. In addition to the atomic formulas of first-order logic mentioned in the previous paragraph, MSO has atomic formulas of the form $X(x)$, saying that the element interpreting the individual variable x is in the set interpreting the set variable X. Furthermore, MSO has quantification over both individual and set variables.

The quantifier rank, denoted $\mathrm{qr}(\psi)$ of a formula ψ is the maximum number of nested quantifiers in ψ. A free variable of a formula ψ is a variable x that does not occur in the scope of a quantifier. The set of free variables of a formula ψ is denoted by $\mathrm{free}(\psi)$. A sentence is a formula without free variables. We write $\psi(x_1, \dots, x_r)$ to indicate that $\mathrm{free}(\psi) \subseteq \{x_1, \dots, x_r\}$. We denote the size of ψ by $||\psi||$. We only work with formulas that have free individual variables, but not free set variables. Given a vector $\overline{x} = \langle x_1, \dots, x_s \rangle$ of variables, a formula $\psi(\overline{x})$ and a structure G, we denote by $\psi(G) = \{\overline{a} \in V^s \mid G \models \psi(\overline{a})\}$ the set of tuples of elements from the universe V of G that can be assigned to the variables $\overline{x}$ to satisfy $\psi(\overline{x})$.

Suppose $\psi(\overline{x}, \overline{y})$ is a formula with two distinguished vectors of free variables $\overline{x}$ of length r and $\overline{y}$ of length s. We call $\psi(\overline{x}, \overline{y})$ a s-ary query with r parameters. Given a structure G, we call $\psi(\overline{a}, G) = \{\overline{b} \in V^s \mid G \models \psi(\overline{a}, \overline{b})\}$ the *output of the query $\psi(\overline{x}, \overline{y})$ with parameter*

$\overline{a}$. We refer to r (resp. s), the length of $\overline{x}$ (resp. y), as the *input length* (resp. the *output length*) of $\psi(\overline{x}, \overline{y})$. Given a weighted structure (G, W), a parametric query $\psi(\overline{x}, \overline{y})$ and a parameter $\overline{a}$, we extend the weight function W to weights of query outputs by defining $W(\psi(\overline{a}, G)) = \Sigma_{\overline{b} \in \psi(\overline{a}, G)} W(\overline{b})$. For a given structure G and a query $\psi(\overline{x}, \overline{y})$, we define $U = \bigcup_{\overline{a} \in V^r} \psi(\overline{a}, G)$ to be the set of *active tuples.*

Watermarking schemes. Suppose $c, d \in \mathbb{N}$. A weighted structure (G, W') is a *c-local distortion* of another weighted structure (G, W) if for all $\overline{b} \in V^s$, $|W'(b) - W(b)| \leq c$. The weighted structure (G, W') is a *d-global distortion* of (G, W) with respect to a query $\psi(\overline{x}, \overline{y})$ if and only if, for all $\overline{a} \in V^r$, $|W'(\psi(\overline{a}, G)) - W(\psi(\overline{a}, G))| \leq d$.

▶ **Definition 2** ([14, 13]). *Given a class of weighted structures $\mathcal{K}$ and a query $\psi(\overline{x}, \overline{y})$, a* watermarking scheme preserving $\psi(\overline{x}, \overline{y})$ *is a pair of algorithms $\mathcal{M}$ (called the marker) and $\mathcal{D}$ (called the detector) along with a function $f : \mathbb{N} \to \mathbb{N}$ and a constant $d \in \mathbb{N}$ such that:*

- *The marker $\mathcal{M}$ takes as input a weighted structure $(G, W) \in \mathcal{K}$ and a mark μ, which is a bit string of length $f(|U|)$, where U is the set of active tuples. It outputs a weighted structure $(G, W_\mu) \in \mathcal{K}$ such that (G, W_μ) is a 1-local and d-global distortion of (G, W) for the query $\psi(\overline{x}, \overline{y})$.*
- *The detector $\mathcal{D}$ is given (G, W), the original structure as input and has access to an oracle that runs queries of the form $\psi(\overline{x}, \overline{y})$ on (G, W_μ). The output of $\mathcal{D}$ is the hidden mark μ.*

Intuitively, the marker takes a bit string and hides it in the database by distorting weights. The detector detects the hidden mark by observing the weights and comparing it with the original weights. The term $f(|U|)$ denotes the length of the bit string that is hidden in the database by the marker. We call a watermarking scheme *scalable* if the function f grows at least as fast as some fractional power asymptotically. For example, the scheme is scalable if $f(n) = \sqrt{n}$ for all n, but not scalable if $f(n) = \log n$ for all n. We will mention later why scalability is important in situations where adversaries try to erase watermarks. Note that the algorithm $\mathcal{D}$ interacts with the marked database (G, W_μ) only through $\psi(\overline{x}, \overline{y})$ queries. Hence, it is not worthwhile distorting the weights of tuples that are not active.

Continuing Example 1, the query $\psi(x)$ given there can be written in First-order as $(\psi(\langle city\rangle, \langle name, salary\rangle) = \texttt{EmployeeTable}(name, city, salary)$. The set of active tuples is $U = \{\langle$John, 10000$\rangle, \langle$Arjun, 20000$\rangle, \langle$Pooja, 15000$\rangle, \langle$Neha, 30000$\rangle, \langle$Padma,20000$\rangle\}$. We can increase the salary of `John` by 1 and decrease the salary of `Pooja` by 1 or vice-versa. Similarly for `Arjun` and `Padma`. This gives 4 different distortions that are 1-local and 0-global. The marker algorithm can take a mark, which is a bit string of length 2, so there are 4 possible marks. The marker can assign these 4 marks to the 4 possible distortions. The detector can observe the changes to the salaries by querying the distorted copy and comparing the results with the original database. The detector can compute the hidden mark by accessing the assignment of the 4 marks to the 4 possible distortions given by the marker. For any instance database of this signature, we can pair off an employee of a city with another employee in the same city and use one such pair to encode one bit of a watermark to be hidden. If there are n active tuples, we can encode $\frac{n}{2}$ bits, assuming that there are at least two employees in each city. For this watermarking scheme, the function f is defined as $f(n) = \frac{n}{2}$ and this is a scalable scheme.

Watermarking schemes can also be put in a context where there are adversaries who know that there is some hidden mark and try to prevent the detector algorithm from working properly, by distorting the database further. Instead of the oracle running queries on (G, W_μ), the queries are run on (G, W'_μ), which is a distortion of (G, W_μ). The detector has to still detect the hidden mark μ correctly. Under some assumptions about the quantity of distortion between (G, W_μ) and (G, W'_μ), watermarking schemes that work in non-adversarial models can be transformed to work in adversarial models; we refer the interested readers to [14, 13]. The correctness of such transformations depend on probabilistic arguments, where scalability helps. With bigger watermarks that are hidden to begin with, there is more room to play around with the distortions introduced by the adversaries.

3 Watermarking schemes

In this section, we prove that scalable watermarking schemes exist for some type of structures. First we prove that if the Gaifman graphs belong to a class of graphs with bounded tree-width, then scalable water marking schemes exist preserving unary MSO queries. Then we prove that if the Gaifman graphs belong to a class of graphs that is closed under minors and that has locally bounded tree-width, then scalable water marking schemes exist preserving unary FO queries.

3.1 MSO Queries on Structures with Bounded Tree-width

3.1.1 Trees, Tree Automata and Clique-width

We begin by reviewing some concepts and known results that are needed.

A binary tree is a $\{S_1, S_2, \preceq\}$-structure, where S_1, S_2 and $\preceq$ are binary relation symbols. A tree $\mathcal{T} = (T, S_1^{\mathcal{T}}, S_2^{\mathcal{T}}, \preceq^{\mathcal{T}})$ has a set of nodes T, a left child relation $S_1^{\mathcal{T}}$ and a right child relation $S_2^{\mathcal{T}}$. Relation $\preceq^{\mathcal{T}}$ stands for the transitive closure of $S_1^{\mathcal{T}} \cup S_2^{\mathcal{T}}$, the tree-order relation. Given a finite alphabet Σ, let $\tau(\Sigma) = \{S_1, S_2, \preceq\} \cup \{P_a | a \in \Sigma\}$ where for all $a \in \Sigma$, P_a is a unary symbol. A Σ-tree is a structure $\mathcal{T} = (T, S_1^{\mathcal{T}}, S_2^{\mathcal{T}}, \preceq^{\mathcal{T}}, (P_a^{\mathcal{T}})_{a\in\Sigma})$, where the restriction $(T, S_1^{\mathcal{T}}, S_2^{\mathcal{T}}, \preceq^{\mathcal{T}})$ is a binary tree and for each $v \in T$ there exists exactly one $a \in \Sigma$ such that $v \in P_a^{\mathcal{T}}$. We denote this unique a by $\sigma^{\mathcal{T}}(v)$. Intuitively, this represents the labelling of nodes by letters from Σ where $\sigma^{\mathcal{T}}(v)$ is the label for the node v. We consider trees with a finite number of pebbles placed on nodes. The pebbles are considered to be distinct: pebble 1 on node v_1 and pebble 2 on node v_2 is not the same as pebble 1 on node v_2 and pebble 2 on node v_1. For some $k \geq 1$, let $\Sigma_k = \Sigma \times \{0, 1\}^k$. This extended alphabet denotes the position of the pebbles in the tree. Suppose $\mathcal{T}$ is a Σ-tree and k pebbles are placed on the nodes $\overline{v} = \langle v_1, \dots, v_k\rangle$. Then $\mathcal{T}_{\overline{v}}$ is the Σ_k-tree with the same underlying tree as $\mathcal{T}$ and $\sigma^{\mathcal{T}_{\overline{v}}}(u) = (\sigma^{\mathcal{T}}(u), \alpha_1, \dots, \alpha_k)$ where $\alpha_i = 1$ if and only if $u = v_i$.

A Σ-tree automaton is a tuple $A = (Q, \delta, F)$ where Q is a set of states and $F \subseteq Q$ are the accepting states. The function $\delta : ((Q \cup \{*\})^2 \times \Sigma) \to Q$ is the transition function, where $*$ is a special symbol not in Q. A run $\rho : T \to Q$ of A on a Σ-tree $\mathcal{T}$ is a function satisfying the following conditions.

- If v is a leaf then $\rho(v) = \delta(*, *, \sigma^{\mathcal{T}}(v))$.
- If v has two children u_1 and u_2, then $\rho(v) = \delta(\rho(u_1), \rho(u_2), \sigma^{\mathcal{T}}(v))$.
- If v has only a left child u then $\rho(v) = \delta(\rho(u), *, \sigma^{\mathcal{T}}(v))$.
- Similarly if v has only a right child.

If v is the root of $\mathcal{T}$, a run ρ of A on $\mathcal{T}$ is an accepting run if $\rho(v) \in F$. A $\Sigma_{(r+s)}$ tree automaton defines a s-ary query with r parameters. We denote by $A(\overline{a}, \mathcal{T}) = \{\overline{b} \in T^s \mid A$ has an accepting run on $\mathcal{T}_{\overline{a}\overline{b}}\}$ the output of the query A on $\mathcal{T}$ with parameter $\overline{a}$. It is well known that MSO queries and tree automata have the same expressive power.

Clique-width. A well-known notion of measuring the similarity of a graph to a tree is its *tree-width.* Many nice properties of trees carry over to classes of structures of bounded tree-width. For our purposes, we use *clique-width*, a related notion. It is well known that a structure of tree-width at most k has clique-width at most 2^k [4].

A k-colored structure is a pair (G, γ) consisting of a structure G and a mapping $\gamma : V \to \{1, \ldots, k\}$. A basic k-colored structure is a k-colored structure (G, γ) where $|V| = 1$ and $R^G = \emptyset$ for all R. We let Γ_k be the smallest class of structures that contain all basic k-colored structures and is closed under the following operations:

- *Union*: take two k-colored structures on disjoint universes and form their union.
- $(i \to j)$ *recoloring, for* $1 \leq i, j \leq k$: take a k-colored structure and recolor all vertices colored i to j.
- $(R, i_1, \ldots i_r)$*-connecting, for every* $r \geq 1$*, every* r*-ary relation symbol* R *and every* $1 \leq i_1, \ldots i_r \leq k$: take a k-colored structure (G, γ) and add all tuples $\langle v_1, \ldots v_r \rangle \in V^r$ with $\gamma(v_j) = i_j$ for $1 \leq j \leq r$ to $\mathcal{R}^G$.

The *clique-width* of a structure G, denoted by $\mathrm{cw}(G)$, is the minimum k such that there exists a k-coloring $\gamma : V \to \{1, \ldots k\}$ such that $(G, \gamma) \in \Gamma_k$.

For every k-colored structure $(G, \gamma) \in \Gamma_k$ we can define a binary, labeled parse-tree in a straightforward way. The leaves of this tree are the elements of G labeled by their color, and each inner node is labeled by the operation it corresponds to. A parse-tree (also called a clique decomposition) of a structure G of clique-width k is a parse tree of some $(G, \gamma) \in \Gamma_k$. For the next lemma, it is important to note that if $\mathcal{T}$ is a parse-tree for a structure G, then $V \subseteq T$.

▶ **Lemma 3** ([12, Lemma 16])**.** *Let* $k \geq 1$*. For every MSO formula* $\varphi(\overline{x})$ *there is a MSO formula* $\tilde{\varphi}(\overline{x})$ *such that for every structure* G *of clique-width* k *and for every parse-tree* $\mathcal{T}$ *of* G *we have* $\varphi(G) = \tilde{\varphi}(\mathcal{T})$*. Furthermore, there are constants* c, d *(only depending on* k *and the signature* τ*) such that* $||\tilde{\varphi}|| \leq c||\varphi||$ *and* $\mathrm{qr}(\tilde{\varphi}) \leq \mathrm{qr}(\varphi) + d$*.*

3.1.2 Watermarking Schemes to Preserve MSO Queries on Structures With Bounded Tree-width

Now we prove that there are scalable watermarking schemes that work for structures from classes with bounded tree-width and preserve a given MSO query. At a high level, the idea is the following: the given MSO query is converted to an equivalent tree automaton. If the number of active elements is large compared to the number of states in the automaton, we can select pairs of elements that can't be distinguished by the automaton. Either both the elements are in the output of the query or none of them are. Hence, distorting one of them by a positive amount and the other one by a negative amount will not contribute to the global distortion.

We begin with the following lemma, which says that if a tree automaton runs on a large tree, we can find large number of pairs of nodes that are "similar" with respect to the automaton. Some of the proofs have been skipped here due to space constraints; they can be found in the full version. A similar result is proved and used in [12] to show that MSO-definable families of sets in graphs of bounded tree-width have bounded Vapnik-Chervonenkis (VC) dimension. The similarity of the following result with that of [12] hints at some possible connections between watermarking schemes and VC dimension.

▶ **Lemma 4.** *Let A be a Σ_{r+1} tree automaton with m states. Let $\mathcal{T}$ be a Σ tree. Suppose $Y \subseteq T$ is a set of nodes of T with at least $2m^m+2$ elements*[2]*. Then, there exists $n = \lfloor \frac{|Y|}{4m^m+4} \rfloor$ pairwise disjoint sets of nodes $V_1, V_2, \ldots, V_n \subseteq T$ and pairs $(b_i, b'_i) \in (V_i \cap Y)^2$ of distinct nodes for all $i \in \{1, \ldots, n\}$ satisfying the following property: for every $\overline{a} = \langle a_1, a_2, \ldots, a_r \rangle \in T^r$ and every $i \in \{1, \ldots, n\}$, if $\{a_1, a_2, \ldots, a_r\} \cap V_i = \emptyset$ then the runs of A on $\mathcal{T}_{\overline{a}b_i}$ and $\mathcal{T}_{\overline{a}b'_i}$ label the tree roots with the same state.*

The following result is proved in [13], but the proof in that paper used a variant of Lemma 4 whose proof has an error. We give a proof with a different constant factor.

▶ **Theorem 5.** *Suppose $\mathcal{K}$ is a class of structures with bounded clique-width. Suppose $\psi(\overline{x}, y)$ is a unary MSO query of input length r, where all the free variables are individual variables. Then, there exists a scalable watermarking scheme preserving $\psi(\overline{x}, y)$ on structures in $\mathcal{K}$.*

Proof. Suppose G is a structure in $\mathcal{K}$, so it has bounded clique-width. From Lemma 3, we get an MSO formula $\tilde{\psi}(\overline{x}, y)$, which can be interpreted on clique decompositions of G to get the same effect as interpreting $\psi(\overline{x}, y)$ on G. We then get an automaton A equivalent to $\tilde{\psi}(\overline{x}, y)$. Let U be the set of active tuples of G for the query $\psi(\overline{x}, y)$. Now we apply Lemma 4, setting $\mathcal{T}$ to be a clique decomposition of G and Y to be the set of active tuples U. We get n pairs $(b_1, b'_1), (b_2, b'_2), \ldots, (b_n, b'_n)$, where n is a constant fraction of $|U|$. Given a weight function W for G and a mark $\mu : \{0,1\}^n$, we define the new weight function W' as follows. We set $(W'(b_i), W'(b'_i)) = (W(b_i)+1, W(b'_i)-1)$ if $\mu(i) = 0$ and $(W'(b_i), W'(b'_i)) = (W(b_i)-1, W(b'_i)+1)$ if $\mu(i) = 1$. For all other elements, W' is same as W. The modified weight function W' has local distortion bounded by 1 by construction. The detector can recover the bits of the mark μ by querying the original and distorted databases and noting the differences in weights assigned to active tuples by W and W'. We will show that it has global distortion bounded by r, the input length of $\psi(\overline{x}, y)$.

Suppose $\overline{a} = \langle a_1, a_2, \ldots, a_r \rangle$ is used as input parameter to the query $\psi(\overline{x}, y)$ on G and (b_i, b'_i) is a pair selected from a set V_i as in Lemma 4. If $\{a_1, \ldots, a_r\} \cap V_i = \emptyset$, then the runs of A on $\mathcal{T}_{\overline{a}b_i}$ and $\mathcal{T}_{\overline{a}b'_i}$ end in the same state. Hence, $b_i \in \psi(\overline{a}, G)$ iff $b'_i \in \psi(\overline{a}, G)$. This means that either both b_i and b'_i are in $\psi(\overline{a}, G)$ or both of them are absent. Hence, the distortion on b_i, b'_i cancel each other, provided $\{a_1, \ldots, a_r\} \cap V_i = \emptyset$. Hence, a pair (b_i, b'_i) may contribute to the global distortion only when $\{a_1, \ldots, a_r\} \cap V_i \neq \emptyset$. Since all the V_i are mutually disjoint and there are at most r elements in $\{a_1, \ldots, a_r\}$, the global distortion is at most r. ◀

Since bounded tree-width implies bounded clique-width, the above result also holds for classes of structures with bounded tree-width.

3.2 FO Queries on Minor Closed Structures with Locally Bounded Tree-width

In this section, we consider structures whose Gaifman graphs belong to a class of graphs that has bounded local tree-width and is closed under minors. We prove that scalable watermarking schemes exist preserving unary first-order queries. We use concepts and techniques from [12] where it is proved that in similar classes of graphs, sets definable by unary first order formulas have bounded VC dimension. It is observed in [12] that this result extends to non-unary formulas. For this extension, [12] uses a generic result from model

[2] A similar result is stated in [13] with $4m$ elements, but there is an error in the proof; see the full version for details.

theory that deals with VC dimension and doesn't use Gaifman graphs. So far, there are no such generic results about watermarking schemes yet. We have not yet found ways to extend our results on watermarking to non-unary queries.

3.2.1 Gaifman's Locality and Locally Bounded Tree-width

First we review some concepts and known results that we use. Given a structure $G = (V, R_1^G, \ldots, R_t^G)$, its *Gaifman graph* is the undirected graph (V, E), where $(v_1, v_2) \in E$ if there is a relation R_i in G and a tuple $\overline{v} \in R_i$ such that v_1 and v_2 appear in $\overline{v}$. The distance between two elements, denoted $d(.,.)$, in a structure is defined to be the shortest distance between them in the Gaifman graph. The distance between two tuples of elements $\overline{v_1}, \overline{v_2}$ is $d(\overline{v_1}, \overline{v_2}) = \min\{d(v_1, v_2) \mid v_1 \in \overline{v_1}, v_2 \in \overline{v_2}\}$. Given $v \in V$, $\rho \in \mathbb{N}$, the ρ-sphere $S_\rho(v)$ is the set $\{v' \mid d(v, v') \leq \rho\}$, and for a tuple $\overline{v}$, $S_\rho(\overline{v}) = \bigcup_{v \in \overline{v}} S_\rho(v)$. We define the ρ-neighborhood around a tuple $\overline{v}$ to be the structure $N_\rho(\overline{v})$ induced on G by $S_\rho(\overline{a})$. The equivalence relation $\approx_\rho$ on tuples of elements is defined as $\overline{a} \approx_\rho \overline{b}$ if $N_\rho(\overline{a}) \approx N_\rho(\overline{b})$ (where $\approx$ denotes isomorphism).

A formula ψ is said to be *local* if there is a number $\rho \in \mathbb{N}$ such that for every G and tuples $\overline{v}_1$ and $\overline{v}_2$ of G, $N_\rho(\overline{v_1}) \approx N_\rho(\overline{v_2})$ implies $G \models \psi(v_1) \iff G \models \psi(v_2)$. This value ρ is then called the *locality radius* of ψ. Gaifman's theorem states that every first-order formula is local. We often annotate a formula ψ with its locality rank r and write it as $\psi^{(r)}$ for the sake of explicitness. Furthermore, $d^{>r}(v_1, v_2)$ is a first-order formula enforcing the distance between v_1 and v_2 to be at least $r + 1$.

▶ **Theorem 6** (Gaifman's locality theorem [10]). *Every First Order formula $\varphi(\overline{x})$ is equivalent to a Boolean combination of the following:*

- *local formulas $\psi^{(\rho)}(\overline{x})$ around $\overline{x}$ and*
- *sentences of the form*

$$\chi = \exists x_1, \ldots, x_s \left(\bigwedge_{i=1}^{s} \alpha^{(\rho)}(x_i) \wedge \bigwedge_{1 \leq i < j \leq s} d^{>2\rho}(x_i, x_j) \right) .$$

Furthermore,

- *The transformation from φ to such a Boolean combination is effective;*
- *If $\mathrm{qr}(\varphi) = q$ and n is the length of $\overline{x}$, then $\rho \leq 7^q$, $s \leq q + n$.*

The (q, k)*-type of $\overline{v}$ in G*, denoted by $\mathrm{tp}_q^G(\overline{v})$, is the set of all first-order formulas $\varphi(x_1, \ldots, x_k)$ of quantifier rank at most q such that $G \models \varphi(\overline{v})$. A (q, k)-type is a maximal consistent set of first-order formulas $\varphi(x_1, \cdots x_k)$ of quantifier rank at most q. Equivalently, a (q, k)-type is the (q, k)-type of some k-tuple $\overline{v}$ in some structure G. For a specific (q, k), there are only finitely many (q, k) types. The number of such types is denoted by $t(q, k)$.

We get the following as a corollary of Gaifman's locality theorem.

▶ **Corollary 7.** *Let $q \in \mathbb{N}$ and $\rho = 7^q$. Let G be a structure and $\overline{a}, \overline{a}' \in V^r$, $\overline{b}, \overline{b}' \in V^s$ such that $\mathrm{tp}_q^G(\overline{a}) = \mathrm{tp}_q^G(\overline{a}')$, $\mathrm{tp}_q^G(\overline{b}) = \mathrm{tp}_q^G(\overline{b}')$, $d(\overline{a}, \overline{b}) \geq 2\rho + 1$ and $d(\overline{a}', \overline{b}') \geq 2\rho + 1$. Then $\mathrm{tp}_q^G(\overline{a}, \overline{b}) = \mathrm{tp}_q^G(\overline{a}', \overline{b}')$.*

Locally Bounded Tree-width. We say that a class of structures $\mathcal{K}$ has *locally-bounded tree-width* if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that given any $G \in \mathcal{K}$, any $v \in V$ and any $r \in \mathbb{N}$, the tree-width of $N_r(x)$ is at most $f(r)$.

Next we recall some properties of class of graphs closed under minors. An edge contraction is an operation which removes an edge from a graph while simultaneously merging the two vertices it used to connect. A graph H is a minor of a graph G if a graph isomorphic to H can be obtained from G by contracting some edges, deleting some edges and deleting some isolated vertices. A class $\mathcal{K}$ of graphs is said to be closed under minors if for every graph G in $\mathcal{K}$ and every minor H of G, H is also in $\mathcal{K}$.

Suppose a class of graphs $\mathcal{K}$ is closed under minors and has locally bounded tree-width (the class of planar graphs is an example). Let G be a graph in $\mathcal{K}$ and let v be an arbitrary vertex in G. For $i \geq 0$, let L_i be the set of all vertices of G whose shortest distance from v is i. For any i, r, the subgraph induced by $\cup_{j=1}^{r} L_{i+j}$ on G has tree-width that depends only on r. See the full version for a proof of this. This idea has been used to design approximation algorithms for hard problems [2, 7].

3.2.2 Watermarking Schemes to Preserve FO Queries on Minor Closed Classes with Locally Bounded Tree-width

Now we prove that there exist watermarking schemes that preserve unary FO queries on classes of structures that are closed under minors and that have locally bounded tree-width. We use Gaifman's locality theorem on the FO query and consider the constituent local queries. Answer to local queries only depend on local neighborhoods of the structure, which have bounded tree-width. We can run automata on them and proceed as in the previous section. We have to be careful that queries run on overlapping neighborhoods don't interfere with each other.

Let $\mathcal{K}$ be a class of structures whose Gaifman graphs belong to a class of graphs with locally bounded tree-width and that is closed under minors, let G be a structure in $\mathcal{K}$ and let $\psi(\overline{x}, y)$ be a unary first-order query. Let q be the rank of $\psi(\overline{x}, y)$ and let ρ be its locality radius. Suppose $U \subseteq V$ is the set of active elements for the query $\psi(\overline{x}, y)$. Let $c \in U$ be an active element such that the set $U_c = \{b \in U \mid \mathrm{tp}_q^G(b) = \mathrm{tp}_q^G(c)\}$ has the maximum cardinality. Due to our choice of c, we get $|U_c| \geq \frac{|U|}{t(q,r+1)}$ (recall that r is the length of $\overline{x}$ and $t(q, r+1)$ is the possible number of $(q, r+1)$-types). We will show that there is a number l that is a constant fraction of $|U|$ such that we can hide any mark $\mu \in \{0,1\}^{\leq l}$. Given a weight function W for G and a mark $\mu \in \{0,1\}^l$, we select l pairs of elements $(b_1, b_1'), (b_2, b_2'), \ldots, (b_l, b_l')$ from U_c and define the new weight function W_μ as follows: $(W_\mu(b_i), W_\mu(b_i')) = (W(b_i) + 1, W(b_i) - 1)$ if $\mu(i) = 1$ and $(W_\mu(b_i), W_\mu(b_i')) = (W(b_i) - 1, W(b_i) + 1)$ if $\mu(i) = 0$. For all other elements, W_μ is same as W. The new weight function is a 1-local distortion of the old one by construction. The difficulty is to ensure that the global distortion is bounded by a constant. We overcome this difficulty by ensuring that b_i and b_i' cannot be distinguished by the query $\psi(\overline{x}, y)$: for almost all $\overline{a} \in V^r$, $b_i \in \psi(\overline{a}, G)$ iff $b_i' \in \psi(\overline{a}, G)$. The following lemma suggests how to select such pairs.

▶ **Lemma 8.** *Suppose $\psi(\overline{x}, y)$ is a query and $\psi_1^{(\rho)}(\overline{x}, y), \psi_2^{(\rho)}(\overline{x}, y), \ldots, \psi_\alpha^{(\rho)}(\overline{x}, y)$ are the local formulas given by Theorem 6 (Gaifman's locality theorem). Suppose G is a structure and $\overline{a} \in V^r, b, b' \in V$ are such that $G \models \psi_i^{(\rho)}(\overline{a}, b)$ iff $G \models \psi_i^{(\rho)}(\overline{a}, b')$ for every $i \in \{1, 2, \ldots, \alpha\}$. Then $b \in \psi(\overline{a}, G)$ iff $b' \in \psi(\overline{a}, G)$.*

Now our goal is to select a large number of pairs (b, b') from U_c such that they cannot be distinguished by any local query $\psi_i^{(\rho)}(\overline{x}, y)$, as assumed in Lemma 8. Let us fix some $k \geq 1$ and apply Lemma 3 to every local query $\psi_i^{(\rho)}(\overline{x}, y)$. We get a MSO formula $\tilde{\psi}_i(\overline{x}, y)$ such that for every structure G' with a parse tree $\mathcal{T}$ of clique-width at most k, $\psi_i^{(\rho)}(G') = \tilde{\psi}_i(\mathcal{T})$.

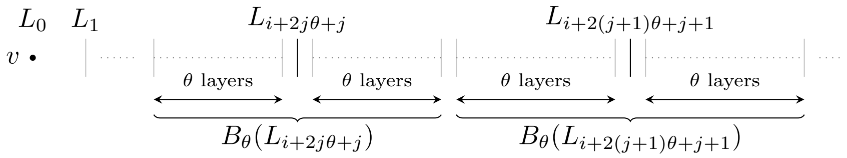


Figure 1 Division of Gaifman's graph of G into Bands and layers.

Our next goal is to identify substructures of G with bounded clique-width. Since we are considering structures of bounded local tree-width, any neighborhood of G of bounded radius has bounded tree-width, hence bounded clique-width.

For the MSO formulas $\tilde{\psi_1}(\overline{x}, y), \tilde{\psi_2}(\overline{x}, y), \ldots, \tilde{\psi_\alpha}(\overline{x}, y)$, let $A_1, A_2, \ldots, A_\alpha$ be the corresponding tree automata. Let A be the tree automaton obtained by applying the usual product construction to $A_1, A_2, \ldots, A_\alpha$ and let m be the number of states in A.

We pick some element $v \in V$ arbitrarily from the universe of G, let $L_0 = \{v\}$, and then define the layer L_i to be the elements of G which are at a distance exactly i from v. This divides the graph into layers $L_0, L_1, L_2, \ldots$. For a layer L_j, define the band $B_{2\rho}(L_j)$ to be the union of the layers $L_{j-2\rho}, L_{j-2\rho+1}, \ldots, L_j, \ldots, L_{j+2\rho-1}, L_{j+2\rho}$. Intuitively, $B_{2\rho}(L_j)$ consists of the layer L_j, 2ρ layers to the left of L_j and 2ρ layers to the right. Let $\theta = (2(r+1)+2)\rho$ and define the band $B_\theta(L_i)$ in an analogous way. For $0 \le i \le 2\theta$, define $\mathcal{L}_i$ to be the set of layers $\{L_i, L_{i+2\theta+1},\ L_{i+4\theta+2}, \ldots\} = \{L_{i+2j\theta+j} \mid j \ge 0\}$. Since there are $2\theta+1$ such sets, it must be the case that there is some $\mathcal{L}_i$ such that $|U_c \cap (\cup \mathcal{L}_i)| \ge \frac{|U_c|}{2\theta+1}$. We denote by $Y_1, Y_2 \ldots$ the layers in this $\cup\mathcal{L}_i$ in increasing order of their distance from L_0. If v is any element in L_j, then $S_{2\rho}(v) \subseteq B_{2\rho}(L_j)$. Notice that by construction, $B_{2\rho}(Y_i) \cap B_{2\rho}(Y_j) = \emptyset = B_\theta(Y_i) \cap B_\theta(Y_j)$ for any $i \neq j$. Refer to Fig. 1 for a visual representation of the bands. The layer L_0 is represented by the single vertex v. The layers $L_{i+2j\theta+j}, L_{i+2(j+1)\theta+j+1}$ are represented by solid vertical lines. Other layers are represented by vertical lines that are grayed out.

In the sequence of layers that we obtained, let $Y'_1, Y'_2, \ldots Y'_\gamma$ be those that contain at least $2m^m+2$ elements from U_c. Let $Y''_1, Y''_2, \ldots, Y''_\delta$ be the layers that contain less than $2m^m+2$ elements from U_c. Let $v''_1, v''_2, \ldots, v''_\delta$ be arbitrarily chosen elements of $Y''_1, Y''_2, \ldots, Y''_\delta$ respectively that are in U_c (we may ignore a particular Y''_i if it does not contain any elements of U_c in it). We will use the set of pairs $M_1 = \{(v''_1, v''_2), \ldots, (v''_{\delta-1}, v''_\delta)\}$ for watermarking.

Next we select watermarking pairs from the layers $Y'_1, Y'_2, \ldots Y'_\gamma$. For each layer Y'_i, let N_i be the substructure induced by the band $B_\theta(Y'_i)$. This is a band of width $2\theta+1$, so its tree-width and hence clique-width (say k) depends only on $2\theta+1$, which in turn depends only on the locality radius ρ and the input length r. Now we can apply Lemma 4 with the tree automaton A and the parse tree $\mathcal{T}$ of N_i of clique width at most k, setting $Y = Y'_i \cap U_c$. We get pairs $(b_{(i,1)}, b'_{(i,1)}), (b_{(i,2)}, b'_{(i,2)}), \ldots, (b_{(i,n)}, b'_{(i,n)})$, where $n = \lfloor \frac{|Y'_i \cap U_c|}{4m^m+4} \rfloor$. We will use the set of pairs $M_2 = \cup_i \cup_j \{(b_{(i,j)}, b'_{(i,j)})\}$ also for watermarking. Again note that all elements in the pairs are in U_c.

▶ **Lemma 9.** *Suppose a watermarking pair $(v''_i, v''_{(i+1)}) \in M_1$ consists of elements from $Y''_i, Y''_{(i+1)}$ respectively. If the tuple $\overline{a} = \langle a_1, \ldots, a_r \rangle$ is such that $\{a_1, \ldots, a_r\} \cap (B_{2\rho}(Y''_i) \cup B_{2\rho}(Y''_{(i+1)})) = \emptyset$, then $v''_i \in \psi(\overline{a}, G)$ iff $v''_{(i+1)} \in \psi(\overline{a}, G)$.*

▶ **Lemma 10.** *Suppose a watermarking pair $(b, b') \in M_2$ was selected from some set V_j (as specified in Lemma 4) of some band N_i. If $\overline{a} = \langle a_1, \ldots, a_r \rangle$ is such that $\{a_1, \ldots, a_r\} \cap V_j = \emptyset$, $b \in \psi(\overline{a}, G)$ iff $b' \in \psi(\overline{a}, G)$.*

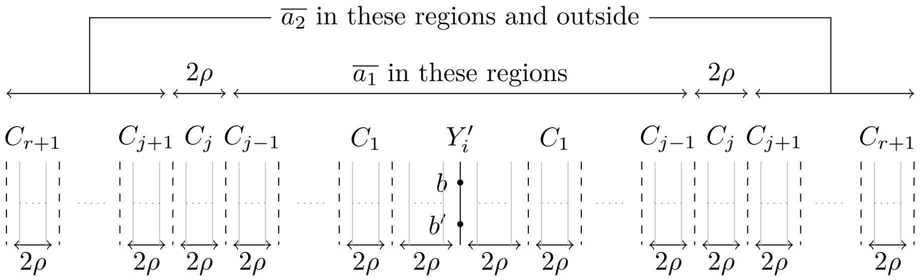

$\overline{a_2}$ in these regions and outside
2ρ
$\overline{a_1}$ in these regions
2ρ
C_{r+1}
C_{j+1} C_j C_{j-1}
C_1
Y_i'
C_1
C_{j-1} C_j C_{j+1}
C_{r+1}
b
b'
2ρ 2ρ 2ρ 2ρ 2ρ 2ρ 2ρ 2ρ 2ρ 2ρ 2ρ 2ρ

Figure 2 The regions $C_1, \ldots, C_{r+1}$ and $\overline{a_1}, \overline{a_2}$ used in Case III of the proof of Lemma 10.

Proof.

Case I: $\{a_1, \ldots, a_r\} \cap B_{2\rho}(Y_i') = \emptyset$. In this case, since b, b' are both on the layer Y_i', we have $S_\rho(\overline{a}) \cap (S_\rho(b) \cup S_\rho(b')) = \emptyset$. Hence we can apply Corollary 7 to infer the result.

Case II: $S_\rho(\overline{a}) \subseteq B_{(2(r+1)+2)\rho}(Y_i')$. In this case, $S_\rho(\overline{a}bb') \subseteq B_{(2(r+1)+2)\rho}(Y_i') = B_\theta(Y_i')$. We selected (b, b') according to Lemma 4, with the tree automaton A running on a parse tree of N_i. Since the tree automaton runs all the automata $A_1, A_2, \ldots, A_\alpha$ simultaneously, we infer that $N_i \models \psi_j^{(\rho)}(\overline{a}, b)$ iff $N_i \models \psi_j^{(\rho)}(\overline{a}, b')$ for every $j \in \{1, 2, \ldots, \alpha\}$. Since $S_\rho(\overline{a}bb') \subseteq B_\theta(Y_i')$, the substructure induced on N_i by $S_\rho(\overline{a}bb')$ is isomorphic to the substructure induced on G by $S_\rho(\overline{a}bb')$. Since $\psi_j^{(\rho)}(\overline{x}, y)$ is a local formula around $\overline{x}, y$ with locality radius ρ, we infer that $N_i \models \psi_j^{(\rho)}(\overline{a}, b)$ iff $G \models \psi_j^{(\rho)}(\overline{a}, b)$ and $N_i \models \psi_j^{(\rho)}(\overline{a}, b')$ iff $G \models \psi_j^{(\rho)}(\overline{a}, b')$ for every $j \in \{1, 2, \ldots, \alpha\}$. Hence, $G \models \psi_j^{(\rho)}(\overline{a}, b)$ iff $G \models \psi_j^{(\rho)}(\overline{a}, b')$ for every $j \in \{1, 2, \ldots, \alpha\}$. We can now apply Lemma 8 to infer the result.

Case III: $\{a_1, \ldots, a_r\} \cap B_{2\rho}(Y_i') \neq \emptyset$ and $\{a_1, \ldots, a_r\} \not\subseteq B_{(2(r+1)+2)\rho}(Y_i')$. In this case, some elements of $\overline{a}$ may be within distance 2ρ from b, b'. Some elements of $\overline{a}$ may be quite far and their ρ neighborhoods may not be included in $B_\theta(Y_i')$. We divide $B_\theta(Y_i') \setminus B_{2\rho}(Y_i')$ into $r+1$ regions $C_1, C_2, \ldots C_{r+1}$. Define $C_1 = B_{4\rho}(Y_i') \setminus B_{2\rho}(Y_i'), C_2 = B_{6\rho}(Y_i') \setminus B_{4\rho}(Y_i')$, etc. Since there are $r+1$ such regions, and only r parameters in $\overline{a}$, there is a region, say C_j that doesn't contain any elements of $\overline{a}$. Let $\overline{a_1}$ be the tuple of elements of $\overline{a}$ that are in $B_{2\rho}(Y_i') \cup C_1 \cup \cdots \cup C_{j-1}$ and let $\overline{a_2}$ consist of the remaining elements of $\overline{a}$. Note that $S_\rho(\overline{a_1}bb') \cap S_\rho(\overline{a_2}) = \emptyset$ (since the region C_j is of width 2ρ, $\overline{a_1}bb'$ are on the inside of this band and $\overline{a_2}$ are on the outside). Refer to Fig. 2 for a visual presentation of $\overline{a_1}, \overline{a_2}$. The layer Y_i' is represented by a solid vertical line, in which b, b' are highlighted. Other layers are represented by vertical lines that are grayed out. Boundaries of regions are represented by dashed vertical lines. Each region consists of 2ρ layers on the left and 2ρ layers on the right. Since the layer C_j doesn't contain any elements of $\overline{a}$, it acts as a buffer between $S_\rho(\overline{a_1}bb')$ and $S_\rho(\overline{a_2})$.

Let the structure H_1 be an isomorphic copy of N_i (which is the substructure induced by $B_\theta(Y_i')$). Since H_1 consists of $2\theta + 1$ layers, the tree-width and hence clique-width of H_1 depends only on $2\theta + 1$. Let H_2 be a disjoint union of at most r spheres of radius at most $2r\rho$, containing an isomorphic copy of $N_\rho(\overline{a_2})$ (details of constructing H_2 are in the full version). The clique-width of H_2 also depends only on r and ρ. Let H be the disjoint union of H_1 and H_2. For the elements $\overline{a_1}bb'$ in N_i, the isomorphism with H_1 will give corresponding elements in H_1; let $h(\overline{a_1}bb')$ be these corresponding elements. Similarly, let $h(\overline{a_2})$ be the elements in H_2 corresponding to $\overline{a_2}$ in $N_\rho(\overline{a_2})$. Let $\mathcal{T}$ be a parse tree of N_i (and so of H_1) and $\mathcal{T}'$ be a parse tree of H_2 of minimum clique-widths, with k being the maximum of these two widths. We obtain a parse tree $\mathcal{T}''$ of H of clique-width at most k by making $\mathcal{T}$ and $\mathcal{T}'$ as subtrees of a new root labeled by the union operation. We selected b, b' according to Lemma 4 with the tree automaton A and parse tree $\mathcal{T}$. We infer that the automaton A labels the roots of $\mathcal{T}_{h(\overline{a_1}b)}$ and $\mathcal{T}_{h(\overline{a_1}b')}$ with the same state. Hence, the automaton A labels the roots of $\mathcal{T}''_{h(\overline{a_1}b\overline{a_2})}$ and $\mathcal{T}''_{h(\overline{a_1}b'\overline{a_2})}$ with the same state (note that $h(\overline{a_1}bb')$ are in $\mathcal{T}$ while $h(\overline{a_2})$ are in $\mathcal{T}'$). Hence, we infer that $H \models \psi_j^{(\rho)}(h(\overline{a}), h(b))$ iff $H \models \psi_j^{(\rho)}(h(\overline{a}), h(b'))$ for every $j \in \{1, 2, \ldots, \alpha\}$. The substructure induced on G by $S_\rho(\overline{a}bb')$ is isomorphic to the substructure induced on H by $S_\rho(h(\overline{a}bb'))$. Since $\psi_j^{(\rho)}(\overline{x}, y)$ is a local formula around $\overline{x}, y$ with locality radius ρ, we infer that $H \models \psi_j^{(\rho)}(h(\overline{a}), h(b))$ iff $G \models \psi_j^{(\rho)}(\overline{a}, b)$ and $H \models \psi_j^{(\rho)}(h(\overline{a}), h(b'))$ iff $G \models \psi_j^{(\rho)}(\overline{a}, b')$. Hence, $G \models \psi_j^{(\rho)}(\overline{a}, b)$ iff $G \models \psi_j^{(\rho)}(\overline{a}, b')$ for every $j \in \{1, 2, \ldots, \alpha\}$. We can now apply Lemma 8 to infer the result. ◀

The technique of considering a small number of layers to get graphs of bounded tree-width was known before [2, 7]. Here, we find ways of using it to bound global distortions and that is the main technical contribution of this paper. Now we state the main result of this sub-section.

▶ **Theorem 11.** *Suppose $\mathcal{K}$ is a class of structures whose Gaifman graphs belong to a class of graphs that is closed under minors and have locally bounded tree-width. Suppose $\psi(\overline{x}, y)$ is a unary first-order query of input length r. Then, there exists a scalable watermarking scheme preserving $\psi(\overline{x}, y)$ on structures in $\mathcal{K}$.*

Proof idea. Every pair in $M_1 \cup M_2$ can hide one bit in the database and $|M_1 \cup M_2|$ is a constant fraction the size of the set of active tuples, as shown in the full version. For a given parameter $\overline{a}$, a pair $(b, b') \in M_1 \cup M_2$ contributes to global distortion only if $\overline{a}$ intersects with some spheres around (b, b'), as proved in Lemma 9 and Lemma 10. The spheres are mutually disjoint, so $\overline{a}$ can intersect with at most r spheres. Hence the global distortion is bounded by r. ◀

4 Conclusion

In [14], there is a transformation of watermarking schemes for non-adversarial models into schemes for adversarial models, under some assumptions. As observed in [13], the same transformation under similar assumptions also work for MSO and FO queries. Hence, our result on FO queries can also use a similar transformation to work on adversarial models.

The difficulty with non-unary queries is that Gaifman graphs don't capture information about active tuples – even if two elements b_1, b_2 appear in the same active tuple, the Gaifman graph may not have an edge between b_1 and b_2. The results on VC dimension use powerful results from model theory [19] or versions of finite Ramsey theorem for hyper graphs [16]. It remains to be seen whether similar results are true for watermarking schemes. It also remains to be seen if the condition on closure under minors can be dropped and watermarking schemes can still be obtained, as shown for VC dimension in [12].

Beginning with graphs of bounded degree, it is now known that for the much bigger class of graphs that are nowhere dense, FO properties can be efficiently decided. It remains to be seen whether results on watermarking schemes can be extended to the class of graphs that are nowhere dense.

We don't know if there are deeper connections between bounded VC dimension and presence of scalable watermarking schemes preserving queries. Some progress is made in [13], where it is shown that unbounded VC dimension doesn't necessarily mean absence of scalable watermarking schemes, but more work is needed in this direction.

References

1 R. Agrawal and J. Kiernan. Watermarking relational databases. In *International Conference on Very Large Databases*, VLDB, 2002.

2 B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41:153–180, 1994.

3 E. F. Codd. Relational completeness of database sublanguages. In *Proceedings of the Sixth Courant Computer Science Symposium on Data Base Systems, Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 65–98, New York, NY, USA, 1972. Prentice-Hall.

4 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000.

5 A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science*, LICS, pages 270–279, 2007.

6 Z. Dvořàk, D. Kràl, and R. Thomas. Deciding first-order properties for sparse graphs. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, FOCS, pages 133–142, 2010.

7 D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27:275–291, 2000.

8 J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model checking. *SIAM Journal on Computing*, 31:113–145, 2001.

9 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

10 H. Gaifman. On Local and Non-Local Properties. In J. Stern, editor, *Proceedings of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. Elsevier, 1982.

11 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding First-Order Properties of Nowhere Dense Graphs. *J. ACM*, 64(3):17:1–17:32, 2017.

12 M. Grohe and G. Turán. Learnability and Definability in Trees and Similar Structures. *Theory of Computing Systems*, 37(1):193–220, January 2004.

13 David Gross-Amblard. Query-preserving Watermarking of Relational Databases and XML Documents. *ACM Trans. Database Syst.*, 36(1):3:1–3:24, 2011.

14 S. Khanna and F. Zane. Watermarking Maps: Hiding Information in Structured Data. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 596–605, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

15 S. Kreutzer. Algorithmic meta-theorems. In J. Esparza, C. Michaux, and C. Steinhorn, editors, *Finite and Algorithmic Model Theory*, pages 177–270. Cambridge University Press, 2011. chapter 5.

16 M.C. Laskowski. Vapnik-Chervonenkis classes of definable sets. *Journal of the London Mathematical Society*, 45(2):377–384, 1992.

17 M. Pilipczuk, S. Siebertz, and S. Toru'nczyk. On the number of types in sparse graphs. In *Proceedings of LICS*, pages 799–808, 2018.

18 D. Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6:505–526, 1996.

19 S. Shelah. Stability, the f.c.p. and superstability. *Annals of Mathematical Logic*, 3:271–362, 1971.

Complexity of Liveness in Parameterized Systems

Peter Chini
TU Braunschweig, Germany
p.chini@tu-braunschweig.de

Roland Meyer
TU Braunschweig, Germany
roland.meyer@tu-braunschweig.de

Prakash Saivasan
TU Braunschweig, Germany
p.saivasan@tu-braunschweig.de

Abstract

We investigate the fine-grained complexity of liveness verification for leader contributor systems. These consist of a designated leader thread and an arbitrary number of identical contributor threads communicating via a shared memory. The liveness verification problem asks whether there is an infinite computation of the system in which the leader reaches a final state infinitely often. Like its reachability counterpart, the problem is known to be NP-complete. Our results show that, even from a fine-grained point of view, the complexities differ only by a polynomial factor.

Liveness verification decomposes into reachability and cycle detection. We present a fixed point iteration solving the latter in polynomial time. For reachability, we reconsider the two standard parameterizations. When parameterized by the number of states of the leader L and the size of the data domain D, we show an $(\mathtt{L}+\mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$-time algorithm. It improves on a previous algorithm, thereby settling an open problem. When parameterized by the number of states of the contributor C, we reuse an $\mathcal{O}^*(2^{\mathtt{C}})$-time algorithm. We show how to connect both algorithms with the cycle detection to obtain algorithms for liveness verification. The running times of the composed algorithms match those of reachability, proving that the fine-grained lower bounds for liveness verification are met.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Problems, reductions and completeness

Keywords and phrases Liveness Verification, Fine-Grained Complexity, Parameterized Systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.37

Related Version A full version of the paper is available as [9] and `https://arxiv.org/abs/1909.12004`.

Acknowledgements We thank Arnaud Sangnier for helpful discussions.

1 Introduction

We study the fine-grained complexity of liveness verification for parameterized systems formulated in the leader contributor model. The model [26, 16] assumes a distinguished leader thread interacting (via a shared memory) with a finite but arbitrary number of indistinguishable contributor threads. The liveness verification problem [14] asks whether there is an infinite computation of the system in which the leader visits a set of final states infinitely often. Fine-grained complexity [13, 10] studies the impact of parameters associated with an algorithmic problem on the problem's complexity like the influence of the contributor size on the complexity of liveness verification. The goal is to develop deterministic algorithms that are provably optimal. We elaborate on the three ingredients of our study.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 37; pp. 37:1–37:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The leader contributor model has attracted considerable attention [26, 16, 14, 31, 17, 23, 7]. From a modeling point of view, a variety of systems can be formulated as anonymous entities interacting with a central authority, examples being client-server applications, resource-management systems, and distributed protocols on wireless sensor networks. From an algorithmic point of view, the model has led to positive surprises. Hague [26] proved decidability of reachability even in a setting where the system components are pushdown automata. La Torre et al. [31] generalized the result to any class of components that satisfies mild assumptions, the most crucial of which being computability of downward closures. As for the complexity, Esparza et al. [16, 17] proved PSPACE-completeness for Hague's model and NP-completeness in the setting where the components are given by finite-state automata. The liveness problem was first studied in [14]. Interestingly, liveness has the same complexity as reachability, it is NP-complete for finite-state systems. Fortin et al. [23] generalized the study to LTL-definable properties and gave conditions for NEXPTIME-completeness.

Fine-grained complexity is a field within parameterized complexity [13, 10]. Parameterized complexity intends to explain the following gap between theory and practice that is observed throughout algorithmics. Despite a high worst-case complexity, tools may have an easy time solving a problem. Parameterized complexity argues that measuring the complexity of a problem in terms of the size of the input, typically denoted by n, is too rough. One should consider further parameters k that capture the shape of the input or the solution sought. Then the gap is due to the fact that tools implement an algorithm running in time $f(k) \cdot \mathit{poly}(n)$. Here, f may be an exponential, but it only depends on the parameter, and that parameter is small in practice. Problems solvable by such an algorithm are called fixed-parameter tractable and belong to the complexity class FPT. Fine-grained complexity is the study of the precise function f that is needed, via upper and lower bound arguments.

The fine-grained complexity of the reachability problem for the leader contributor model was studied in our previous work [7]. We assumed that the components are finite state and considered two parameterizations. When parameterized by the size of the contributors $\mathtt{C}$, we showed that reachability can be solved in time $\mathcal{O}^*(2^{\mathtt{C}})$. The notation $\mathcal{O}^*$ suppresses polynomial factors in the running time. Interestingly, this is the best one can hope for. An algorithm with a subexponential dependence on $\mathtt{C}$, to be precise an algorithm running in time $2^{o(\mathtt{C})}$, would contradict the so-called exponential time hypothesis (ETH). The ETH [28] is a standard hardness assumption in parametrized complexity that is used to derive relative lower bounds. The second parameterization is by the size of the leader $\mathtt{L}$ and the size of the data domain $\mathtt{D}$. We gave an algorithm running in time $(\mathtt{LD})^{\mathcal{O}(\mathtt{LD})}$. Interestingly, the lower bound is only $2^{o((\mathtt{L}+\mathtt{D}) \cdot \log(\mathtt{L}+\mathtt{D}))}$. Being away a quadratic factor in the exponent means a substantial gap for a deterministic algorithm.

In the present paper, we study the fine-grained complexity of the liveness verification problem. We assume finite-state components and consider the same parameterization as for reachability. The surprise is in the parameterization by $\mathtt{L}$ and $\mathtt{D}$. We give an algorithm running in time $(\mathtt{L}+\mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$. This matches the lower bound and closes the gap for reachability. When parameterized by the size of the contributors, we obtain an $\mathcal{O}^*(2^{\mathtt{C}})$ algorithm.

To explain the algorithms, note that a live computation decomposes into a prefix and an accepting cycle. Finding prefixes is a matter of reachability. We show how to combine reachability algorithms with a cycle detection to obtain algorithms that find live computations. The resulting algorithms will run in time $\mathcal{O}(\mathit{Reach}(\mathtt{L}, \mathtt{D}, \mathtt{C}) \cdot \mathit{Cycle}(\mathtt{L}, \mathtt{D}, \mathtt{C}))$ where $\mathit{Reach}(\mathtt{L}, \mathtt{D}, \mathtt{C})$ denotes the running time of the invoked reachability algorithm and $\mathit{Cycle}(\mathtt{L}, \mathtt{D}, \mathtt{C})$ that of the cycle detection. This result allows for considering reachability and cycle detection separately.

Our first main contribution is an algorithm for reachability when $\mathtt{L}$ and $\mathtt{D}$ are given as parameters. It runs in time $(\mathtt{L}+\mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$ and significantly improves upon the $(\mathtt{LD})^{\mathcal{O}(\mathtt{LD})}$-time algorithm from [7]. Moreover, it is optimal in the fine-grained sense. It closes the gap between upper and lower bound. The algorithm works over sketches of computations. A sketch is valid if there is an actual computation corresponding to it. In [7], we performed a single validity check for each sketch. Here, we show that valid sketches can be build up inductively from small sketches. To this end, we interleave validity checks with compression phases. Our algorithm is a dynamic programming on small sketches, exploiting the inductive approach.

Our second main result is an algorithm for detecting cycles. We show that the problem is actually solvable in polynomial time. Technically, we employ a characterization of cycles via (certain) SCC decompositions of the contributor automaton. These decompositions can be computed by a fixed point iteration invoking Tarjan's algorithm [35] in polynomial time.

Since $\mathit{Cycle}(\mathtt{L}, \mathtt{D}, \mathtt{C})$ is polynomial, liveness has the same complexity as reachability also in the fine-grained sense. With the above result, we obtain the mentioned algorithms for liveness by composing the reachability algorithms with the cycle detection.

Related Work. The parameterized complexity has also been studied for other verification problems. Farzan and Madhusudan [18] consider the problem of predicting atomicity violations. Depending on the synchronization, they obtain an efficient fine-grained algorithm resp. prove an FPT-algorithm unlikely. In [15], the authors give an efficient (fine-grained) algorithm for the problem of checking TSO serializability. In [5], we studied the fine-grained complexity of bounded context switching [33], including lower bounds on the complexity. In [7], we gave a parameterized analysis of the bounded write-stage restriction, a generalization of bounded context switching [2]. The problem turns out to be hard for different parameterizations, and has a large number of hard instances. In a series of papers [20, 19, 36], Fernau et al. studied FPT-algorithms for problems from automata theory.

Related to leader contributor systems are broadcast networks (ad-hoc networks) [34, 12]. These consist of an arbitrary number of finite-state contributors that communicate via message passing. There is no leader. This has an impact on the complexity of safety [11, 24] and liveness [6, 3] verification, which drops from NP (leader contributor systems) to P.

More broadly, the verification of parameterized systems is an active field of research [4]. Prominent approaches are well-structuredness arguments [1, 21] and cut-off results [25]. Well-structuredness means the transition relation is monotonic wrt. a well-quasi ordering on the configurations, a combination that leads to surprising decidability results. A cut-off is a bound on the size of system instances such that correctness of the bounded instances entails correctness of all instances. Our algorithm uses different techniques. We give a reduction from liveness to reachability combined with a polynomial-time cycle check. Reductions from liveness to reachability or safety are recently gaining popularity in verification [29, 32, 27]. For reachability, we then rely on techniques from parameterized complexity [13, 10], namely identifying combinatorial objects to iterate over and dynamic programming.

2 Leader Contributor Systems and the Liveness Problem

We introduce leader contributor systems and the leader contributor liveness problem of interest following [26, 16, 14]. Moreover, we give a short introduction to fine-grained complexity. For standard textbooks, we refer to [22, 10, 13].

Leader Contributor Systems. A *leader contributor system* consists of a designated leader thread communicating with a number of identical contributor threads via a shared memory. Formally, the system is a tuple $\mathcal{S} = (D, a^0, P_L, P_C)$ where D is the finite domain of the shared memory and $a^0 \in D$ is the initial memory value. The leader P_L and the contributor P_C

are abstractions of concrete threads making visible the interaction with the memory. They are defined as finite state automata over the alphabet $Op(D) = \{!a, ?a \mid a \in D\}$ of memory operations. Here, $!a$ denotes a write of a to the memory, $?a$ denotes a read of a. The leader is given by the tuple $P_L = (Op(D), Q_L, q^0_L, \delta_L)$ where Q_L is the set of states, $q^0_L \in Q_L$ is the initial state, and $\delta_L \subseteq Q_L \times (Op(D) \cup \{\varepsilon\}) \times Q_L$ is the transition relation. We extend the relation to words in $Op(D)^*$ and usually write $q \xrightarrow{w}_L q'$ for $(q, w, q') \in \delta_L$. The contributor is defined similarly, by $P_C = (Op(D), Q_C, q^0_C, \delta_C)$.

The possible interactions of a thread with the memory depend on the current memory value and the internal state of the thread. To keep track of this information, we use *configurations.* These are tuples of the form $(q, a, pc) \in CF^t = Q_L \times D \times Q^t_C$. Here, pc is a vector storing the current state of each contributor, and there are $t \in \mathbb{N}$ contributors participating in the computation. The number of participating contributors can be arbitrary, but will be fixed throughout the computation. Therefore, the set of all configurations is given by $CF = \bigcup_{t\in\mathbb{N}} CF^t$. A configuration is called *initial* if it is of the form (q^0_L, a^0, pc^0) where $pc^0(i) = q^0_C$ for each $i \in [1..t]$. We use projections to access the components of a configuration. Let π_L and π_D denote the projections to the leader state resp. the memory content, $\pi_L((q, a, pc)) = q$ and $\pi_D((q, a, pc)) = a$. The map π_C projects a configuration to the set of contributor states present in pc, $\pi_C((q, a, pc)) = \{pc(i) \mid i \in [1..t]\}$.

The current configuration of $\mathcal{S}$ may change due to an interaction with the memory or an internal transition. We capture such changes by a labeled transition relation among configurations, $\rightarrow \subseteq CF \times (Op(D) \cup \{\varepsilon\}) \times CF$. It contains transitions induced by the leader and by the contributor. We focus on the former. If there is a write $q \xrightarrow{!b}_L q'$ of the leader, we get $(q, a, pc) \xrightarrow{!b} (q', b, pc)$. Similarly, a read $q \xrightarrow{?a}_L q'$ induces $(q, a, pc) \xrightarrow{?a} (q', a, pc)$. Note that the current memory value has to match the read symbol. An internal transition $q \xrightarrow{\varepsilon}_L q'$ yields $(q, a, pc) \xrightarrow{\varepsilon} (q', a, pc)$. For the transitions induced by the contributors, let $pc(i) = p$ and $pc' = pc[i = p']$, meaning $pc'(i) = p'$ and pc' coincides with pc in all other components. A transition $p \xrightarrow{!b/?a/\varepsilon}_C p'$ yields $(q, a, pc) \xrightarrow{!b/?a/\varepsilon} (q, b/a, pc')$, like for the leader. Note that transitions are only defined among configurations involving the same number of contributors. It is convenient to assume that the leader never writes $!a$ and immediately reads $?a$ again. In this case, we could replace the corresponding read transition by ε.

The transition relation $\rightarrow$ is generalized to words, denoted by $c \xrightarrow{w} c'$ with $w \in Op(D)^*$. We call such a sequence a *computation* of $\mathcal{S}$. We also write $c \rightarrow^* c'$ if there is a word w with $c \xrightarrow{w} c'$, and $c \rightarrow^+ c'$ if w has length at least 1. An *infinite computation* is a sequence $\sigma = c^0 \rightarrow c^1 \rightarrow \ldots$ of infinitely many transitions. We call it *initialized* if c^0 is an initial configuration. Since σ involves infinitely many configurations but the set Q_L is finite, there are states of the leader that occur infinitely often along the computation. We denote the set of these states by $\mathrm{Inf}(\sigma) = \{q \in Q_L \mid \exists^\infty i : q = \pi_L(c^i)\}$.

Leader Contributor Liveness. The *leader contributor liveness problem* is the task of deciding whether the leader satisfies a liveness specification while interacting with a number of contributors. Formally, given a leader contributor system $\mathcal{S} = (D, a^0, P_L, P_C)$ and a set of final states $F \subseteq Q_L$ encoding the specification, the problem asks whether there is an initialized infinite computation σ such that the leader visits F infinitely often along σ. Since F is finite, this is equivalent to $\mathrm{Inf}(\sigma) \cap F \neq \emptyset$. In this case, σ is called a *live computation.*

Leader Contributor Liveness (LCL)
Input: A leader contributor system $\mathcal{S} = (D, a^0, P_L, P_C)$ and final states $F \subseteq Q_L$.
Question: Is there an infinite initialized computation σ such that $\mathrm{Inf}(\sigma) \cap F \neq \emptyset$?

Fine-Grained Complexity. The problem LCL is known to be NP-complete [14]. Despite its hardness, it may still admit efficient deterministic algorithms the running times of which depend exponentially only on certain parameters. To find parameters that allow for the construction of such algorithms, one examines the *parameterized complexity* of LCL. Note that the name does not refer to parameterized systems. It stems from measuring the complexity not only in the size of the input but also in the mentioned parameters.

Let Σ be an alphabet. Unlike in classical complexity theory where we consider problems over Σ^*, a *parameterized problem* P is a subset of $\Sigma^* \times \mathbb{N}$. Inputs to P are pairs (x, k) with the second component k being referred to as the *parameter*. Problem P is called *fixed-parameter tractable* if it admits a deterministic algorithm deciding membership in P for pairs (x, k) in time $f(k) \cdot |x|^{\mathcal{O}(1)}$. Here, f is a computable function that only depends on k. Since f usually dominates the polynomial, the running time of the algorithm is denoted by $\mathcal{O}^*(f(k))$.

While finding an upper bound for the function f amounts to coming up with an efficient algorithm, lower bounds on f are obtained relative to hardness assumptions. One of the standard assumptions is the *exponential time hypothesis* (ETH) [28]. It asserts that 3-SAT cannot be solved in time $2^{o(n)}$ where n is the number of variables in the input formula. The lower bound is transported to the problem of interest via a reduction from 3-SAT. Then, f cannot drop below a certain bound unless ETH fails. It is a task of *fine-grained complexity* to find the *optimal* function f, where upper and lower bound match.

We conduct fine-grained complexity analyses for two parameterizations of LCL. First, we consider LCL($\mathtt{L}, \mathtt{D}$), the parameterization by the number of states in the leader $\mathtt{L}$ and the size of the data domain $\mathtt{D}$. We show an $(\mathtt{L} + \mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$-time algorithm, matching the lower bound for LCL from [7]. The second parameterization LCL($\mathtt{C}$) is by the number of states of the contributor $\mathtt{C}$. We give an algorithm running in time $\mathcal{O}^*(2^{\mathtt{C}})$. It also matches the known lower bound [7]. Therefore, both algorithms are optimal in the fine-grained sense. The parameterizations LCL($\mathtt{L}$) and LCL($\mathtt{D}$) are unlikely to be fixed-parameter tractable. These problems are hard for W[1], a complexity class comprising intractable problems [7].

3 Dividing Liveness along Interfaces

A live computation naturally decomposes into a prefix and a cycle. This means that solving LCL amounts to finding both, a prefix computation and a cyclic computation. However, we need to guarantee that the computations can be linked. The prefix should lead to a configuration that the cycle loops on. Since there are infinitely many configurations, we introduce the finite domain of interfaces. An interface abstracts a configuration to its leader state, memory value, and set of contributor states. Hence, an interface can be seen as a summary of those configurations that are suitable for linking prefix and cycle.

Our algorithm to solve LCL works as follows. We start a reachability algorithm for the leader contributor model on the final states that the live computation should visit. After a modification, the algorithm outputs all interfaces witnessing prefixes to those states. Let *Reach*($\mathtt{L}, \mathtt{D}, \mathtt{C}$) denote the running time of the reachability algorithm. We show that the obtained set of interfaces will be of size at most *Reach*($\mathtt{L}, \mathtt{D}, \mathtt{C}$). We iterate over the interfaces and pass each to a cycle detection which works over interfaces instead of configurations. If a cycle was found, a live computation exists. Let *Cycle*($\mathtt{L}, \mathtt{D}, \mathtt{C}$) be the time needed for a single cycle detection. Then, the running time of the algorithm can be estimated as follows.

▶ **Theorem 1.** LCL *can be solved in time* $\mathcal{O}(\mathit{Reach}(\mathtt{L}, \mathtt{D}, \mathtt{C}) \cdot \mathit{Cycle}(\mathtt{L}, \mathtt{D}, \mathtt{C}))$.

The first step in proving Theorem 1 is to decompose live computations into prefixes and cycles. To be precise, we aim for a decomposition where the cycle is saturated in the sense that the initial configuration already contains all contributor states that will be encountered along the cycle. Knowing these states in advance eases technical arguments when finding cycles in Section 5. Formally, a cyclic computation $\tau = c \to^* c$ is called *saturated* if for each configuration c' in τ, we have $\pi_C(c') \subseteq \pi_C(c)$. We write $c \to^*_{sat} c$ for a saturated cycle. The following lemma yields the desired decomposition. If not stated otherwise, proofs and details for the current section are provided in the full version of the paper.

▶ **Lemma 2.** *There is an infinite initialized computation σ with $\mathrm{Inf}(\sigma) \cap F \neq \emptyset$ if and only if there is a finite initialized computation $c^0 \to^* c \to^+_{sat} c$ with $\pi_L(c) \in F$.*

We would like to decompose LCL into finding prefix and cycle. But we need to ensure that the found computations can be linked at an explicit configuration. For avoiding the latter, we introduce interfaces. An *interface* is a triple $I = (S, q, a) \in \mathcal{P}(Q_C) \times Q_L \times D$ consisting of a set of contributor states S, a state of the leader q, and a memory value a. A configuration c *matches* the interface I if $\pi_C(c) = S$, $\pi_L(c) = q$, and $\pi_D(c) = a$. We denote this by $I(c)$, interpreting I as a predicate. The set of interfaces is denoted by IF. The following lemma shows that the notion allows for decomposing LCL. We can search for prefixes and cycles separately. The lemma provides the arguments needed to complete the proof of Theorem 1.

▶ **Lemma 3.** *Let $I \in \mathrm{IF}$. There is a computation $c^0 \to^* c \to^+_{sat} c$ with $I(c)$ if and only if there are computations $d^0 \to^* d$ and $f \to^+_{sat} f$ with $I(d) \wedge I(f)$.*

In the following, we turn to our main contributions. We present algorithms for reachability and cycle detection and obtain precise values for *Reach*($\mathtt{L}, \mathtt{D}, \mathtt{C}$) and *Cycle*($\mathtt{L}, \mathtt{D}, \mathtt{C}$). Further, we modify the reachability algorithms to output interfaces. Then we invoke Theorem 1 to derive algorithms for LCL. The first problem that we consider is finding prefixes.

Leader Contributor Reachability (LCR)
Input: A leader contributor system $\mathcal{S} = (D, a^0, P_L, P_C)$ and final states $F \subseteq Q_L$.
Question: Is there an initialized computation $c^0 \to^* c$ with $\pi_L(c) \in F$?

The problem LCR is NP-complete [16]. Its complexity *Reach*($\mathtt{L}, \mathtt{D}, \mathtt{C}$) depends on the parameterization. There are two standard parameterizations [7, 8]: LCR($\mathtt{L}, \mathtt{D}$) and LCR($\mathtt{C}$).

For the parameterization by $\mathtt{L}$ and $\mathtt{D}$, we present an algorithm solving LCR($\mathtt{L}, \mathtt{D}$) in time $(\mathtt{L} + \mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$. The algorithm solves an open problem [7] by matching the known lower bound: unless ETH fails, LCR cannot be solved in time $2^{o((\mathtt{L}+\mathtt{D})\cdot \log(\mathtt{L}+\mathtt{D}))}$. The algorithm and its modification for obtaining interfaces are presented in Section 4.

▶ **Theorem 4.** LCR($\mathtt{L}, \mathtt{D}$) *can be solved in time* $(\mathtt{L} + \mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$.

For LCR($\mathtt{C}$), we modify the reachability algorithm from [7, 8] so that it outputs interfaces that witness prefixes. We recall the result on the complexity of the algorithm.

▶ **Theorem 5** ([7, 8])**.** LCR($\mathtt{C}$) *can be solved in time* $\mathcal{O}(2^{\mathtt{C}} \cdot \mathtt{C}^4 \cdot \mathtt{L}^2 \cdot \mathtt{D}^2)$.

The second task to solve LCL is detecting cycles. We formalize the problem. It takes an interface and asks for a saturated cycle on a configuration that matches the interface.

Saturated Cycle (CYC)
Input: A leader contributor system $\mathcal{S} = (D, a^0, P_L, P_C)$ and an interface $I \in \mathrm{IF}$.
Question: Is there a computation $c \to^+_{sat} c$ with $I(c)$?

We present an algorithm solving CYC in polynomial time. Key to the algorithm is a fixed point iteration over certain subgraphs of the contributor. Details are postponed to Section 5.

▶ **Theorem 6.** CYC *can be solved in time* $\mathcal{O}(D^2 \cdot (C^2 + L^2 \cdot D^2))$.

The theorem shows that $\mathit{Cycle}(\mathtt{L}, \mathtt{D}, \mathtt{C})$ is polynomial. Hence, by Theorem 1, we obtain that LCL can be solved in time $\mathcal{O}^*(\mathit{Reach}(\mathtt{L}, \mathtt{D}, \mathtt{C}))$. This means that liveness verification and safety verification in the leader contributor model only differ by a polynomial factor. Taking the precise values for $\mathit{Reach}(\mathtt{L}, \mathtt{D}, \mathtt{C})$ into account, Theorem 1 yields the following.

▶ **Corollary 7.** $\mathsf{LCL}(L, D)$ *can be solved in time* $(L + D)^{\mathcal{O}(L+D)}$.

▶ **Corollary 8.** $\mathsf{LCL}(C)$ *can be solved in time* $\mathcal{O}(2^C \cdot L \cdot D^2 \cdot (L \cdot C^4 + D \cdot C^2 + L^2 \cdot D^3))$.

For the latter result, we are actually more precise in determining the time complexity than stated in Theorem 1. Both obtained algorithms are optimal. They match the corresponding lower bounds for LCL that carry over from reachability [7]. Unless ETH fails, LCL cannot neither be solved in time $2^{o((\mathtt{L}+\mathtt{D}) \cdot \log(\mathtt{L}+\mathtt{D}))}$ nor in time $2^{o(\mathtt{C})}$.

4 Reachability Parameterized by Leader and Domain

We present the algorithm for $\mathsf{LCR}(\mathtt{L}, \mathtt{D})$. It runs in time $(\mathtt{L} + \mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$ and therefore proves Theorem 4. Moreover, with the results from Section 3 and 5, the algorithm can be utilized for solving LCL in time $(\mathtt{L} + \mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$. Like in [7], the algorithm relies on a notion of witnesses. These are sketches of computations. A witness is valid if there is an actual computation following the sketch. Validity can be checked in polynomial time.

The algorithm from [7] iterates over all witnesses and tests validity for each. Hence, the time complexity of the algorithm is proportional to $(\mathtt{LD})^{\mathcal{O}(\mathtt{LD})}$, the number of considered witnesses. Key to our new algorithm is the fact that we can restrict to so-called short witnesses. These are sketches of loop-free computations. We show that validity of witnesses can be checked inductively from validity of short witnesses. We exploit the inductivity by a dynamic programming. It runs in time proportional to $(\mathtt{L} + \mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$, the number of short witnesses. This yields the desired complexity as stated in Theorem 4.

4.1 Witnesses and Validity

We introduce witnesses and recall the notion of validity. Afterwards, we elaborate on the main idea of our new algorithm: restricting to short witnesses for checking validity.

Intuitively, a witness is a compact way to represent computations of a leader contributor system. From a computation, a witness only stores the actions of the leader and the positions where memory symbols were written by a contributor for the first time. We call these positions *first writes*. From such a position on, we can assume an unbounded supply of the corresponding memory symbol. There is always a copy of a contributor waiting to provide it.

Formally, a *witness* is a triple $x = (w, q, \sigma)$. The word $w = (q_1, a_1)(q_2, a_2) \dots (q_n, a_n)$ represents the run of the leader. It is a sequence from $(Q_L \times (D \uplus \{\bot\}))^*$, containing leader states potentially combined with a memory value. The state $q \in Q_L$ is the target of the leader run. First-write positions are specified by $\sigma : [1..k] \to [1..n]$, a monotonically increasing map where $k \leq \mathtt{D}$. The number of first-write positions k is called the *order* of x. We denote it by $\mathit{ord}(x) = k$. Moreover, we use Wit for the set of all witnesses. A witness $x = (w, q, \sigma) \in \mathit{Wit}$ is called *initialized* if w begins in the initial state q_L^0 of the leader automaton.

If a witness corresponds to an actual computation, we call it valid. This means, the witness encodes a proper run of the leader and moreover, the first writes along the run can be provided by the contributors. Since the definition of witnesses only specifies first-write positions but not values, we need the notion of first-write sequences. The latter will allow for the definition of validity.

A *first-write sequence* is a sequence of data values $\beta \in D^{\leq \mathsf{D}}$ that are all different. Formally, $\beta_i \neq \beta_j$ for $i \neq j$. We use FW to denote the set of all those sequences. Given a witness $x = (w, q, \sigma)$, we define its validity with respect to a first-write sequence β of length $ord(x)$. For being valid, x has to be *leader valid along* β and *contributor valid along* β. We make both notions more precise. Details regarding this section including formal definitions are available in the full version of the paper.

Leader Validity. The witness is *leader valid along* β if w encodes a run of the leader that reaches state q. Reading during the run is restricted to symbols from β: the ℓ-th symbol β_ℓ is available for reading once the run arrives at position $\sigma(\ell)$. Formally, the encoding depends on the memory values a_i. If $a_i \neq \bot$, the leader has a transition $q_i \xrightarrow{!a_i}_L q_{i+1}$. If $a_i = \bot$, the leader either has an ε-transition or reads a symbol available at position i, from the set $S_\beta(i) = \{\beta_\ell \mid \sigma(\ell) \leq i\}$. We use $\mathrm{LValid}_\beta(x)$ to indicate that x is leader valid along β.

Contributor Validity. The witness is *contributor valid along* β if the contributors can provide the first writes for w in the order indicated by σ. Let us focus on the i-th first write β_i. Providing β_i is a question of reachability of the set $Q_i = \{p \mid \exists p' : p \xrightarrow{!\beta_i}_C p'\}$ in the contributor automaton. More precise, we need a contributor that reaches Q_i while reading only symbols available along w. This means that reading is restricted to earlier first writes and symbols written by the leader during w up to position $\sigma(i)$.

Let $Expr(x, \beta_1 \dots \beta_{i-1})$ be the language of available reads. We say that x is *valid for the i-th first write of* β if Q_i is reachable by a contributor while reading is restricted to $Expr(x, \beta_1 \dots \beta_{i-1})$. We use $\mathrm{CValid}^i_\beta(x)$ to indicate this validity. If x is valid for all first writes, it is *contributor valid along* β. Formally, $\mathrm{CValid}_\beta(x) = \bigwedge_{i \in [1..ord(x)]} \mathrm{CValid}^i_\beta(x)$.

With leader and contributor validity in place, we can define x to be *valid along* β if $\mathrm{LValid}_\beta(x) \wedge \mathrm{CValid}_\beta(x)$. Again, we use predicate notation. We write $\mathrm{Valid}_\beta(x)$ if x is valid along β. Validity of a witness along a first-write sequence can be checked in polynomial time.

▶ **Lemma 9.** *Let $x \in Wit$ and $\beta \in$ FW. $\mathrm{Valid}_\beta(x)$ can be evaluated in polynomial time.*

The algorithm from [7] iterates over witnesses and invokes Lemma 9 to check validity. The following lemma proves the correctness: validity indicates the existence of a computation.

▶ **Lemma 10.** *Let $q \in Q_L$. There is an initialized computation $c^0 \to^* c$ with $\pi_L(c) = q$ if and only if there is an initialized $x = (w, q, \sigma) \in Wit$ and a $\beta \in FW$ so that $\mathrm{Valid}_\beta(x)$.*

For obtaining a tractable algorithm, we would like to restrict to short witnesses when checking validity. These are witnesses encoding a loop-free run of the leader. The following two observations are crucial to our development.

Leader validity can be checked inductively on short witnesses. A witness x can be written as a product $x = x_1 \times x_2 \times \dots \times x_{k+1}$ of smaller witnesses. Each x_i encodes that part of the leader run of x happening between two first-write positions $\sigma(i-1)$ and $\sigma(i)$. The *witness concatenation* × appends these runs. Each x_i can assumed to be a short witness. There is no need for recording loops of the leader between first writes. We can cut them out.

Assume $y = x_1 \times \dots \times x_i$ encodes a proper run ρ of the leader that reads from the available first writes $\beta_1, \dots, \beta_{i-1}$. Formally, $\mathrm{LValid}_{\beta_1 \dots \beta_{i-1}}(y)$. Then, leader validity of $y \times x_{i+1}$ along $\beta_1 \dots \beta_i$ mainly depends on the newly added witness x_{i+1}. The reason is that we prolong ρ, a run of the leader that was already verified. All that we have to remember from ρ is where it ends. This means that we can shrink y to a short witness. We consecutively cut out loops from the leader, denoted by *Shrink**, until we obtain a loop free witness. Formally, if $\mathrm{LValid}_{\beta_1\dots\beta_{i-1}}(y)$ holds true, we have the equality

$$\mathrm{LValid}_{\beta_1\dots\beta_i}(y \times x_{i+1}) = \mathrm{LValid}_{\beta_1\dots\beta_i}(\mathit{Shrink}^*(y) \times x_{i+1}).$$

Hence, checking leader validity can be restricted to (concatenations of) short witnesses.

Like leader validity, we can restrict contributor validity to short witnesses. The main reason is that testing validity for the i-th first write only requires limited knowledge about earlier first writes. As long as we guarantee that earlier first writes can be provided along a run of the leader, we do not have to keep track of their precise positions anymore. This means that we can shrink the run when testing validity for the i-th first write.

Assume that $y = x_1 \times \dots \times x_i$ is known to be contributor valid. Formally, $\mathrm{CValid}_{\beta_1\dots\beta_{i-1}}(y)$ is true. Note that the first writes considered in y are $\beta_1, \dots, \beta_{i-1}$. We want to check contributor validity of $y \times x_{i+1}$. Since there is only one new first write that we add, namely β_i, we have to evaluate $\mathrm{CValid}^i_{\beta_1\dots\beta_i}(y \times x_{i+1})$. Satisfying contributor validity means that β_i can be provided along $y \times x_{i+1}$ assuming that $\beta_1, \dots, \beta_{i-1}$ were already provided. In fact, it is not important where these earlier first writes appeared exactly. We just need the fact that after y, they can assumed to be there. This allows for shrinking y and forgetting about the precise positions of the earlier first writes. Formally, if $\mathrm{CValid}_{\beta_1\dots\beta_{i-1}}(y)$, we have

$$\mathrm{CValid}^i_{\beta_1\dots\beta_i}(y \times x_{i+1}) = \mathrm{CValid}^i_{\beta_1\dots\beta_i}(\mathit{Shrink}^*(y) \times x_{i+1}).$$

In the next section, we turn the above observations into a recursive definition of validity for short witnesses. The recursion only involves short witnesses of lower order. Since the number of these is bounded by $(\mathtt{L}+\mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$, we can employ a dynamic programming that checks validity of short witnesses in time proportional to their number.

4.2 Algorithm and Correctness

Before we can formulate the recursion, we need to introduce short witnesses and a concatenation operator on the same. A *short witness* is a witness $z = (w, q, \sigma) \in \mathit{Wit}$ where the leader states in $w = (q_1, a_1) \dots (q_n, a_n)$ are all distinct. We use Wit^{sh} to denote the set of all short witnesses. Moreover, let $\mathrm{Ord}(k)$ denote the set of those short witnesses that are of order k.

Let $x = (w, q, \sigma) \in \mathrm{Ord}(i)$ and $y = (w', q', \sigma') \in \mathrm{Ord}(j)$ be two short witnesses. Assume that the first state in w' is q, meaning that y starts with the target state of x. Then, the *short concatenation* of x and y is defined to be the short witness $x \otimes y = \mathit{Shrink}^*(x \times y) \in \mathrm{Ord}(i+j)$.

The price to pay for the smaller number of short witnesses is a more expensive check for validity. Rather than checking validity once for each short witness, we build them up by a recursion along the order, and check validity for each composition. Let z be a short witness. If $\mathit{ord}(z) = 0$, there are no first-write positions. Only leader validity is important:

$$\mathrm{Valid}^{sh}_{\varepsilon}(z) = \mathrm{LValid}_{\varepsilon}(z).$$

For a short witness z of order $k+1$, we define validity along $\beta = \beta_1 \dots \beta_{k+1} \in \mathrm{FW}$ by

$$\mathrm{Valid}^{sh}_{\beta}(z) = \bigvee_{\substack{x \in \mathrm{Ord}(k)\\ y \in \mathrm{Ord}(1)}} [z = x \otimes y] \wedge \mathrm{LValid}_{\beta}(x \times y) \wedge \mathrm{CValid}^{k+1}_{\beta}(x \times y) \wedge \mathrm{Valid}^{sh}_{\beta'}(x).$$

Here $\beta' = \beta_1 \dots \beta_k$ is the prefix of β where the last element is omitted.

The idea behind the recursion is to cut off the last first write β_{k+1}, check its validity, and recurse on the remaining part. To this end, z is decomposed into two short witnesses $x \in \mathrm{Ord}(k)$ and $y \in \mathrm{Ord}(1)$. Intuitively, x is the compression of a larger witness that is already known to be valid and y is the short witness responsible for the last first write. By our considerations above, we already know that it suffices to check validity for β_{k+1} with x instead of its expanded form. These are the evaluations $\mathrm{LValid}_{\beta}(x \times y)$ and $\mathrm{CValid}_{\beta}^{k+1}(x \times y)$. To guarantee validity along β', we recurse on $\mathrm{Valid}_{\beta'}^{sh}(x)$.

The following lemma shows the correctness of the recursion. Using Lemma 10, we can work with short witnesses to discover computations in the given leader contributor system.

▶ **Lemma 11.** *Let $q \in Q_L$ and $\beta \in \mathrm{FW}$. There is an $x = (w, q, \sigma) \in \mathit{Wit}$ with $\mathrm{Valid}_{\beta}(x)$ if and only if there is an $z = (w', q, \sigma') \in \mathit{Wit}^{sh}$ with $\mathrm{Valid}_{\beta}^{sh}(z)$. In this case, $\mathit{init}(x) = \mathit{init}(z)$.*

Note that in the lemma, $\mathit{init}(x)$ refers to the first state of w. Similarly for z.

It remains to give the algorithm. For each first-write sequence β and each short witness z, we compute $\mathrm{Valid}_{\beta}^{sh}(z)$ by a dynamic programming. To this end, we maintain a table indexed by first-write sequences and short witnesses. An entry for $\beta \in \mathrm{FW}$ and $z \in \mathit{Wit}^{sh}$ is computed as follows. Let $|\beta| = \mathit{ord}(z) = k$. We iterate over all short witnesses $x \in \mathrm{Ord}(k-1), y \in \mathrm{Ord}(1)$ and check whether $z = x \otimes y$ holds. If so, we compute $\mathrm{LValid}_{\beta}(x \times y) \wedge \mathrm{CValid}_{\beta}^{k}(x \times y)$ and look up the value of $\mathrm{Valid}_{\beta'}^{sh}(x)$ in the table. Details on the precise complexity are presented in the full version of the paper.

▶ **Proposition 12.** *The set of all valid short witnesses can be computed in time $(\mathtt{L}+\mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$.*

It is left to explain how interfaces can be obtained from the algorithm. From a valid short witness, target state and last memory value can be read off. Contributor states can be obtained by synchronizing the contributor along the witness. This takes polynomial time. Details can be found in the full version of the paper.

5 Finding Cycles in Polynomial Time

We give an efficient algorithm solving CYC in time $\mathcal{O}(\mathtt{D}^2 \cdot (\mathtt{C}^2 + \mathtt{L}^2 \cdot \mathtt{D}^2))$. This proves Theorem 6. The algorithm relies on a characterization of cycles in terms of stable SCC decompositions. These are decompositions of the contributor automaton into strongly connected subgraphs that are stable in the sense that they write exactly the symbols they intend to read. With a fixed point iteration, we show how to find stable SCC decompositions in the mentioned time.

Our algorithm is technically simple. It relies on a fixed point iteration calling Tarjan's algorithm [35] to obtain SCC decompositions. Hence, the algorithm is easy to implement and shows that stable SCC decompositions are the ideal structure for detecting cycles. Moreover, we can modify the algorithm to detect cycles where the leader necessarily makes a move.

We also discovered that cycles can be detected by a non-trivial polynomial-time reduction to the problem of finding cycles in dynamic graphs. Although the latter can be solved in polynomial time [30], the obtained algorithm for CYC does not admit an efficient polynomial-time complexity. The reason is that the algorithm in [30] repeatedly solves linear programs that grow large due to the reduction. Compared to this method, our algorithm is more efficient and technically simpler due to being tailored to the actual problem.

5.1 From Saturated Cycles to Stable SCC decompositions

We characterize cycles in terms of stable SCC decompositions. These are decompositions of the contributor automaton that can provide themselves with all the symbols that a cycle along this structure may read. For the definition, we generalize properties of a fixed cycle to the fact that a saturated cycle exists. We link the latter with an alphabet Γ, a variable for the set of reads in a saturated cycle. Then we define stable SCC decompositions depending on Γ. Hence, the search for a cycle amounts to finding a Γ with a stable SCC decomposition.

Throughout the section, we fix an interface $I = (S, q, a)$ and a saturated cycle $\tau = c \rightarrow^{+}_{sat} c$ with $I(c)$. We assume that the set $\mathrm{Writes}(\tau) = \{b \in D \mid d \xrightarrow{!b} d' \in \tau\}$ is non-empty, τ contains at least one write. If τ contains only reads, then either a contributor or the leader run in an $?a$-loop, a cycle which is easy to detect. We generalize two properties of τ.

Property 1: Strongly connectedness. Considering the saturated cycle τ, we can observe how the current state of a particular contributor P changes over time. Assume P starts in a state p and visits a state p' during τ. Since it runs along the cycle, the contributor will eventually move from p' back to p again. This means that in the contributor automaton, there is a path from p to p' and vice versa. Phrased differently, p and p' are strongly connected.

To make this notion more precise, we define a subgraph of the contributor automaton. Intuitively, it is the restriction of P_C to the states and transitions visited along τ. Rather than defining it for a single computation τ, we generalize to a set of *enabled reads* $\Gamma \subseteq D$. The directed graph $G_S(\Gamma) = (S, E(\Gamma))$ has as vertices the contributor states S and as edges the set $E(\Gamma)$. The latter are transitions of P_C between states in S that are either reads enabled by Γ or writes of arbitrary symbols. Formally, we have

$$(p, p') \in E(\Gamma) \text{ if } p \xrightarrow{?b}_C p' \text{ with } b \in \Gamma \text{ or } p \xrightarrow{!b}_C p' \text{ with } b \in D.$$

For the cycle $\tau = c \rightarrow^{+}_{sat} c$, the induced graph is $G_S(\Gamma)$ where $\Gamma = \mathrm{Writes}(\tau)$. With the graph in place, we can define our notion of strongly connected states.

▶ **Definition 13.** *Let $p, p' \in S$ be two states and $\Gamma \subseteq D$. We say that p and p' are* strongly Γ-connected *if p and p' are strongly connected in the graph $G_S(\Gamma)$.*

Like the classical notion, the above definition generalizes to sets. We say that a set $V \subseteq S$ is *strongly Γ-connected* if each two states in V are strongly Γ-connected.

The saturated cycle τ runs along the SCC decomposition of its induced graph $G_S(\Gamma)$. Following a particular contributor P in τ, we collect the visited states in a set $S_P \subseteq S$. Then, S_P is strongly Γ-connected and thus contained in an inclusion maximal strongly connected set, an SCC of $G_S(\Gamma)$. Hence, the contributors in τ stay within SCCs of the graph. We associate with τ the SCC decomposition. Again, we generalize to a given alphabet.

Let $\Gamma \subseteq D$ and $V \subseteq S$ strongly Γ-connected. We call V a *strongly Γ-connected component* (Γ-SCC) if it is inclusion maximal. The latter means that for each $V \subseteq V'$ with V' strongly Γ-connected, we already have $V = V'$. We consider the unique partition of S into Γ-SCCs. Note that by a partition, we mean a collection $(S_1, \ldots, S_\ell)$ of pairwise disjoint subsets of S such that $S = \bigcup_{i \in [1..\ell]} S_i$. The order of a partition is not important for our purpose.

▶ **Definition 14.** *The partition of S into Γ-SCCs is called* Γ-SCC decomposition *of S.*

We denote the Γ-SCC decomposition by $\mathit{SCCdcmp}_S(\Gamma)$. It consists of the vertices of the SCC decomposition of $G_S(\Gamma)$. Hence, we can obtain it from an application of Tarjan's algorithm [35], a fact that becomes important when computing $\mathit{SCCdcmp}_S(\Gamma)$ in Section 5.2.

Property 2: Stability. Let $\mathit{SCCdcmp}_S(\Gamma) = (S_1, \ldots, S_\ell)$ be the Γ-SCC decomposition associated with the saturated cycle τ. The writes in τ can be linked with the S_i. If a write occurs between states $p, p' \in S_i$, we associate it with the set S_i. The writes of the leader all occur on a cyclic computation $q \rightarrow^{*}_{L} q$. The point of assigning writes to sets is the following. Writes that belong to a set can occur on a cycle through a set of the decomposition.

We generalize from τ to a given alphabet $\Gamma \subseteq D$. Let $\mathit{SCCdcmp}_S(\Gamma) = (S_1, \dots, S_\ell)$ be the Γ-SCC decomposition of S. The *writes of the decomposition* is the set of all symbols that occur as writes either between the states of S_i or in a cycle $q \rightarrow_L^* q$ on the leader while preserving the memory content a. Formally, we define the writes to be the union $\mathrm{Writes}(S_1, \dots, S_\ell) = \mathrm{Writes}_C(S_1, \dots, S_\ell) \cup \mathrm{Writes}_L(S_1, \dots, S_\ell)$ where

$$\mathrm{Writes}_C(S_1, \dots, S_\ell) = \{b \mid p \xrightarrow{!b}_C p' \text{ with } p, p' \in S_i\} \text{ and}$$
$$\mathrm{Writes}_L(S_1, \dots, S_\ell) = \{b \mid \exists u, v : (q, a) \xrightarrow{u.!b.v}_{L'} (q, a)\}.$$

Here, $\rightarrow_{L'}$ denotes the transition relation of the automaton $P_{L'}$, a restriction of the leader P_L to reads within $\mathrm{Writes}_C(S_1, \dots, S_\ell)$. The automaton also keeps track of the memory content. We define $P_{L'} = (Op(D), Q_L \times D, (q_L^0, a^0), \delta_{L'})$ with the transitions

$$\begin{aligned} &(s, b) \xrightarrow{!b'}_{L'} (s', b') && \text{if } s \xrightarrow{!b'}_L s', \\ &(s, b) \xrightarrow{?b}_{L'} (s', b) && \text{if } s \xrightarrow{?b}_L s' \text{ and } b \in \mathrm{Writes}_C(S_1, \dots, S_\ell), \\ &(s, b) \xrightarrow{\varepsilon}_{L'} (s, b') && \text{if } b' \in \mathrm{Writes}_C(S_1, \dots, S_\ell). \end{aligned}$$

The last transitions change the memory content due to a write of a contributor.

The following lemma states that writes behave monotonically. This fact will become important in Section 5.2. We provide a proof in the full version of the paper.

▶ **Lemma 15.** *Let* $\Gamma \subseteq \Gamma' \subseteq D$. *We have* $\mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma)) \subseteq \mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma'))$.

During the cycle τ, reads are always preceded by corresponding writes. Hence, the writes of the Γ-SCC decomposition, where $\Gamma = \mathrm{Writes}(\tau)$, provide all symbols needed for reading. In fact, we have $\mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma)) \supseteq \Gamma$. The following definition generalizes this property.

▶ **Definition 16.** *Let* $\Gamma \subseteq D$. *The* Γ*-SCC decomposition* $\mathit{SCCdcmp}_S(\Gamma)$ *of* S *is called* stable *if it provides* Γ *as its writes, meaning* $\mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma)) = \Gamma$.

Note that the definition asks for equality instead of inclusion. The reason is that we can express stability as a fixed point of a suitable operator. This will be essential in Section 5.2.

Characterization. The following proposition characterizes the existence of saturated cycles via stable SCC decompositions. It is a major step towards the polynomial-time algorithm.

▶ **Proposition 17.** *There is a saturated cycle* $\tau = c \rightarrow_{sat}^{+} c$ *with* $I(c)$ *if and only if there exists a non-empty subset* $\Gamma \subseteq D$ *such that* $\mathit{SCCdcmp}_S(\Gamma)$ *is stable.*

Proof. Assume the existence of a saturated cycle τ. Our candidate set is $\Gamma = \mathrm{Writes}(\tau)$. We already argued above that $\mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma)) \supseteq \Gamma$. If equality holds, $\mathit{SCCdcmp}_S(\Gamma)$ is stable and Γ is the set we are looking for. Otherwise, we have $\mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma)) \supsetneq \Gamma$.

In the latter case, we consider $\Gamma' = \mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma))$ instead of Γ. Since $\Gamma' \supseteq \Gamma$, we can apply Lemma 15 and obtain that $\mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma'))$ contains Γ'.

Iterating this process yields a sequence of sets $(\Gamma_i)_i$ that is strictly increasing, $\Gamma_i \subsetneq \Gamma_{i+1}$, and that satisfies $\mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma_i)) \supseteq \Gamma_i$. The sequence is finite since $\Gamma_i \subseteq D$ for all i. Hence, there is a last set Γ_d which necessarily fulfills $\mathrm{Writes}(\mathit{SCCdcmp}_S(\Gamma_d)) = \Gamma_d$.

For the other direction, we need to construct a saturated cycle from a set Γ with stable SCC decomposition. Idea and formal proof are given in the full version of the paper. ◀

5.2 Computing Stable SCC decompositions

The search for a saturated cycle reduces to finding an alphabet Γ with a stable SCC decomposition. Following the definition of stability, we can express Γ as a fixed point that can be computed by a Kleene iteration [37] in polynomial time. We define the suitable operator. It acts on the powerset lattice $\mathcal{P}(D)$ and for a given set X, it computes the writes of the X-SCC decomposition. Formally, it is defined by

$$\mathrm{Writes}_{SCC}(X) = \mathrm{Writes}(\mathit{SCCdcmp}_S(X)).$$

The operator is monotone and can be evaluated in polynomial time.

▶ **Lemma 18.** *For $X \subseteq X'$ subsets of D, we have $\mathrm{Writes}_{SCC}(X) \subseteq \mathrm{Writes}_{SCC}(X')$. Moreover, $\mathrm{Writes}_{SCC}(X)$ can be computed in time $\mathcal{O}(\mathtt{D} \cdot (\mathtt{C}^2 + \mathtt{L}^2 \cdot \mathtt{D}^2))$.*

Monotonicity follows from Lemma 15. For the evaluation, let X be given. We apply Tarjan's algorithm on $G_S(X)$ to compute the X-SCC decomposition $\mathit{SCCdcmp}_S(X)$. This takes linear time. It is left to compute the writes $\mathrm{Writes}(\mathit{SCCdcmp}_S(X))$. For details on the computation and the precise complexity we refer to the full version.

The following lemma states that the non-trivial fixed points of the operator Writes_{SCC} are precisely the sets with a stable SCC decomposition. Hence, searching for a cycle reduces to searching for a fixed point.

▶ **Lemma 19.** *For $\Gamma \neq \emptyset$ we have, $\Gamma = \mathrm{Writes}_{SCC}(\Gamma)$ if and only if $\mathit{SCCdcmp}_S(\Gamma)$ is stable.*

Correctness immediately follows from the definition of stability. For finding a suitable set Γ, we employ a Kleene iteration to compute the greatest fixed point of Writes_{SCC}. It starts from $\Gamma = D$, the top element of the lattice. At each step, it evaluates $\mathrm{Writes}_{SCC}(\Gamma)$ by invoking Lemma 18. This takes time $\mathcal{O}(\mathtt{D} \cdot (\mathtt{C}^2 + \mathtt{L}^2 \cdot \mathtt{D}^2))$. Termination is after at most $\mathtt{D}$ steps since at least one element is removed from the set Γ each iteration. Hence, the time to compute the greatest fixed point of Writes_{SCC} is $\mathcal{O}(\mathtt{D}^2 \cdot (\mathtt{C}^2 + \mathtt{L}^2 \cdot \mathtt{D}^2))$.

6 Conclusion

We studied the fine-grained complexity of LCL, the liveness verification problem for leader contributor systems. To this end, we first decomposed LCL into the reachability problem LCR and the cycle detection CYC. We focused on the complexity of LCR. While an optimal $\mathcal{O}^*(2^{\mathtt{C}})$-time algorithm for $\mathsf{LCR}(\mathtt{C})$ was already known, we presented an algorithm solving $\mathsf{LCR}(\mathtt{L}, \mathtt{D})$ in time $(\mathtt{L} + \mathtt{D})^{\mathcal{O}(\mathtt{L}+\mathtt{D})}$. The algorithm is optimal in the fine-grained sense and therefore solves an open problem. It is a dynamic programming based on a notion of valid short witnesses. Moreover, we showed how to modify both algorithms for LCR so that they are compatible with a cycle detection and can be used in algorithms solving LCL.

Further, we determined the complexity of CYC. We presented an efficient fixed point iteration running in time $\mathcal{O}(\mathtt{D}^2 \cdot (\mathtt{C}^2 + \mathtt{L}^2 \cdot \mathtt{D}^2))$. It is based on a notion of stable SCC decompositions and invokes Tarjan's algorithm to find them. The result shows that LCL and LCR admit the same fine-grained complexity.

References

1 P. A. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. In *LICS*, pages 160–170. IEEE, 1993.

2 M. F. Atig, A. Bouajjani, K. N. Kumar, and P. Saivasan. On Bounded Reachability Analysis of Shared Memory Systems. In *FSTTCS*, volume 29 of *LIPIcs*, pages 611–623. Schloss Dagstuhl, 2014.

3 N. Bertrand, P. Fournier, and A. Sangnier. Playing with Probabilities in Reconfigurable Broadcast Networks. In *FOSSACS*, volume 8412 of *LNCS*, pages 134–148. Springer, 2014.

4 R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

5 P. Chini, J. Kolberg, A. Krebs, R. Meyer, and P. Saivasan. On the Complexity of Bounded Context Switching. In *ESA*, volume 87, pages 27:1–27:15. Schloss Dagstuhl, 2017.

6 P. Chini, R. Meyer, and P.Saivasan. Liveness in Broadcast Networks. In *NETYS*, 2019.

7 P. Chini, R. Meyer, and P. Saivasan. Fine-Grained Complexity of Safety Verification. In *TACAS*, volume 10806 of *LNCS*, pages 20–37. Springer, 2018.

8 P. Chini, R. Meyer, and P. Saivasan. Fine-Grained Complexity of Safety Verification. *CoRR*, abs/1802.05559, 2018. arXiv:1802.05559.

9 P. Chini, R. Meyer, and P. Saivasan. Complexity of Liveness in Parameterized Systems. *CoRR*, abs/1909.12004, 2019. arXiv:1909.12004.

10 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.

11 G. Delzanno, A. Sangnier, R. Traverso, and G. Zavattaro. On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks. In *FSTTCS*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl, 2012.

12 G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized Verification of Ad Hoc Networks. In *CONCUR*, volume 6269 of *LNCS*, pages 313–327. Springer, 2010.

13 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

14 A. Durand-Gasselin, J. Esparza, P. Ganty, and R. Majumdar. Model Checking Parameterized Asynchronous Shared-Memory Systems. In *CAV*, volume 9206 of *LNCS*, pages 67–84. Springer, 2015.

15 C. Enea and A. Farzan. On Atomicity in Presence of Non-atomic Writes. In *TACAS*, volume 9636 of *LNCS*, pages 497–514. Springer, 2016.

16 J. Esparza, P. Ganty, and R. Majumdar. Parameterized Verification of Asynchronous Shared-Memory Systems. In *CAV*, pages 124–140, 2013.

17 J. Esparza, P. Ganty, and R. Majumdar. Parameterized Verification of Asynchronous Shared-Memory Systems. *JACM*, 63(1):10:1–10:48, 2016.

18 A. Farzan and P. Madhusudan. The Complexity of Predicting Atomicity Violations. In *TACAS*, volume 5505 of *LNCS*, pages 155–169. Springer, 2009.

19 H. Fernau, P. Heggernes, and Y. Villanger. A multi-parameter analysis of hard problems on deterministic finite automata. *JCSS*, 81(4):747–765, 2015.

20 H. Fernau and A. Krebs. Problems on Finite Automata and the Exponential Time Hypothesis. In *CIAA*, volume 9705 of *LNCS*, pages 89–100. Springer, 2016.

21 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1-2):63–92, 2001.

22 F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.

23 M. Fortin, A. Muscholl, and I. Walukiewicz. Model-Checking Linear-Time Properties of Parametrized Asynchronous Shared-Memory Pushdown Systems. In *CAV*, volume 8044 of *LNCS*, pages 155–175. Springer, 2017.

24 P. Fournier. *Parameterized verification of networks of many identical processes*. PhD thesis, University of Rennes 1, 2015.

25 S. M. German and A. P. Sistla. Reasoning about Systems with Many Processes. *JACM*, 39(3):675–735, 1992.

26 M. Hague. Parameterised Pushdown Systems with Non-Atomic Writes. In *FSTTCS*, volume 13 of *LIPIcs*, pages 457–468. Schloss Dagstuhl, 2011.

27 M. Hague, R. Meyer, S. Muskalla, and M. Zimmermann. Parity to Safety in Polynomial Time for Pushdown and Collapsible Pushdown Systems. In *MFCS*, volume 117 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl, 2018.

28 R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *JCSS*, 62(2):367–375, 2001.

29 I. V. Konnov, M. Lazic, H. Veith, and J. Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL*, pages 719–734. ACM, 2017.

30 S. R. Kosaraju and G. F. Sullivan. Detecting Cycles in Dynamic Graphs in Polynomial Time (Preliminary Version). In *STOC*, pages 398–406. ACM, 1988.

31 S. La Torre, A. Muscholl, and I. Walukiewicz. Safety of Parametrized Asynchronous Shared-Memory Systems is Almost Always Decidable. In *CONCUR*, volume 42 of *LIPIcs*, pages 72–84. Schloss Dagstuhl, 2015.

32 O. Padon, J. Hoenicke, G. Losa, A. Podelski, M. Sagiv, and S. Shoham. Reducing liveness to safety in first-order logic. *PACMPL*, 2(POPL):26:1–26:33, 2018.

33 S. Qadeer and J. Rehof. Context-Bounded Model Checking of Concurrent Software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

34 A. Singh, C. R. Ramakrishnan, and S. A. Smolka. Query-Based Model Checking of Ad Hoc Network Protocols. In *CONCUR*, volume 5710 of *LNCS*, pages 603–619. Springer, 2009.

35 R. E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SICOMP*, 1(2):146–160, 1972.

36 T. Wareham. The Parameterized Complexity of Intersection and Composition Operations on Sets of Finite-State Automata. In *CIAA*, volume 2088 of *LNCS*, pages 302–310. Springer, 2000.

37 G. Winskel. *The formal semantics of programming languages - an introduction*. Foundation of computing series. MIT Press, 1993.

Greibach Normal Form for ω-Algebraic Systems and Weighted Simple ω-Pushdown Automata

Manfred Droste
Institut für Informatik, Universität Leipzig, Germany
droste@informatik.uni-leipzig.de

Sven Dziadek
Institut für Informatik, Universität Leipzig, Germany
dziadek@informatik.uni-leipzig.de

Werner Kuich
Institut für Diskrete Mathematik und Geometrie, Technische Unversität Wien, Austria
werner.kuich@tuwien.ac.at

Abstract

In weighted automata theory, many classical results on formal languages have been extended into a quantitative setting. Here, we investigate weighted context-free languages of infinite words, a generalization of ω-context-free languages (Cohen, Gold 1977) and an extension of weighted context-free languages of finite words (Chomsky, Schützenberger 1963). As in the theory of formal grammars, these weighted languages, or ω-algebraic series, can be represented as solutions of mixed ω-algebraic systems of equations and by weighted ω-pushdown automata.

In our first main result, we show that mixed ω-algebraic systems can be transformed into *Greibach normal form*. Our second main result proves that *simple ω-reset pushdown automata* recognize all ω-algebraic series that are a solution of an ω-algebraic system in Greibach normal form. Simple reset automata do not use ϵ-transitions and can change the stack only by at most one symbol. These results generalize fundamental properties of context-free languages to weighted languages.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Quantitative automata; Theory of computation → Grammars and context-free languages

Keywords and phrases Weighted omega-Context-Free Grammars, Algebraic Systems, Greibach Normal Form, Weighted Automata, omega-Pushdown Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.38

Funding *Sven Dziadek*: supported by DFG Research Training Group 1763 (QuantLA).
Werner Kuich: partially supported by Austrian Science Fund (FWF): grant no. I1661 N25.

1 Introduction

Context-free languages provide a fundamental concept for programming languages in computer science. In order to model quantitative properties, already in 1963, Chomsky and Schützenberger [3] introduced weighted context-free languages. The theory of weighted pushdown automata developed quickly; for background, we refer the reader to the survey [19] and the books [21, 20, 16, 10]. In 1977, Cohen and Gold [4] investigated context-free languages of infinite words. Weighted pushdown automata on infinite words were studied more recently by Ésik and Kuich [14].

The goal of this paper is the investigation of weighted context-free languages and weighted pushdown automata on infinite words. As in [20, 16], the weighted context-free languages of finite and infinite words are described by solutions of mixed ω-algebraic systems of equations. In our first main result, we show that these systems can be transformed into a Greibach normal form. In the literature, Greibach normal forms, central for context-free languages of

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 38; pp. 38:1–38:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

finite words, have been established for ω-context-free languages (of infinite words), see [4], and also for algebraic systems of equations for series over finite words [20, 16]; this latter result is employed in our proof. Hence here we extend these classical results to a weighted version for infinite words.

In our second main result, we consider weighted *simple* pushdown automata. These automata do not use ϵ-transitions and utilize only three simple stack commands: popping a symbol, pushing a symbol or leaving the stack unaltered; moreover, it is only possible to read the topmost stack symbol by popping it. Observe that together with the restriction of not allowing ϵ-transitions, these restrictions for the actions on the stack are non-trivial. In our second main result we show that these weighted *simple* pushdown automata still recognize the weighted ω-context-free languages that are a solution of weighted ω-context-free grammars in Greibach normal form. Our proof uses two ingredients. First, weighted ω-pushdown automata are expressively equivalent to mixed ω-algebraic systems of equations, see [8, 9]. Secondly, we apply a recent corresponding expressive equivalence result for weighted simple pushdown automata on finite words from [7] to construct the required weighted simple ω-pushdown automata.

We believe the model of weighted simple ω-pushdown automata to be very natural. Similar expressivity equivalence results in the unweighted case hold for context-free languages of finite words, hidden in a proof by Blass and Gurevich [1], and also for ω-context-free languages, see [6].

After the preliminaries in the next section, Sections 3 and 4 contain our results on the Greibach normal form. Sections 5 and 6 describe weighted simple pushdown automata.

2 Preliminaries

For the convenience of the reader, we recall definitions and results from Ésik, Kuich [16].

A semiring S is called *complete* if it has "infinite sums" (i) that are an extension of the finite sums, (ii) that are associative and commutative and (iii) that satisfy the distribution laws (see Conway [5], Eilenberg [12], Kuich [19]).

A semiring S equipped with an additional unary star operation $^* : S \to S$ is called a *starsemiring*. In complete semirings for each element a, the *star* a^* of a is defined by

$$a^* = \sum_{j \geq 0} a^j .$$

Hence, each complete semiring is a starsemiring, called a *complete starsemiring*.

A semiring is called *continuous* if it is ordered, each directed subset has a least upper bound and addition and multiplication preserves the least upper bound of directed sets. Any continuous semiring is complete. See Ésik, Kuich [16] for background.

Suppose that S is a semiring and V is a commutative monoid written additively. We call V a (left) S-semimodule if V is equipped with a (left) action

$$S \times V \;\to\; V, \qquad (s, v) \;\mapsto\; sv$$

subject to the following rules:

$$s(s'v) = (ss')v\,, \quad (s + s')v = sv + s'v\,, \quad s(v + v') = sv + sv'\,,$$
$$1v = v\,, \quad 0v = 0\,, \quad s0 = 0\,,$$

for all $s, s' \in S$ and $v, v' \in V$. If V is an S-semimodule, we call (S, V) a *semiring-semimodule pair*.

Suppose that (S, V) is a semiring-semimodule pair such that S is a starsemiring and S and V are equipped with an omega operation $^\omega : S \to V$. Then we call (S, V) a *starsemiring-omegasemimodule pair.* A semiring-semimodule pair (S, V) is called *complete* if S is a complete semiring, V is a complete monoid and the left action of the semimodule is distributive; moreover, it is required that it has "infinite products" mapping infinite sequences over S to V such that the product (i) can be partitioned, (ii) can be extended from the left and (iii) is distributive (see Ésik, Kuich [17]).

Suppose that (S, V) is complete. Then we define

$$s^* = \sum_{i\geq 0} s^i \qquad \text{and} \qquad s^\omega = \prod_{i\geq 1} s\,,$$

for all $s \in S$. This turns (S, V) into a starsemiring-omegasemimodule pair. Observe that, if (S, V) is a complete semiring-semimodule pair, then $0^\omega = 0$.

A *star-omega semiring* is a semiring S equipped with unary operations * and $^\omega : S \to S$. A star-omega semiring S is called *complete* if (S, S) is a complete semiring-semimodule pair, i.e., if S is complete and is equipped with an infinite product operation that satisfies the three conditions stated above. A complete star-omega semiring S is called *continuous* if the semiring S is continuous.

For the definition of quemirings, we refer the reader to [16], page 110. Here we note that a quemiring T is isomorphic to a quemiring $S \times V$ determined by the semiring-semimodule pair (S, V); this is an algebraic structure with an addition given componentwise and a multiplication given by semiring multiplication in the first component and a semidirect product type addition in the second component (since S acts on V); cf. Elgot [13], Ésik, Kuich [16], page 109. Also, one can define a natural star operation on $S \times V$, see [16].

For an alphabet Σ, we call mappings r of Σ^* into S *series.* The collection of all such series r is denoted by $S\langle\langle\Sigma^*\rangle\rangle$. We call the set $\mathrm{supp}(r) = \{w \mid (r, w) \neq 0\}$ the support of a series r. We denote by $S\langle\Sigma\rangle$, $S\langle\{\epsilon\}\rangle$ and $S\langle\Sigma \cup \{\epsilon\}\rangle$ the series with support in Σ, $\{\epsilon\}$ and $\Sigma \cup \{\epsilon\}$, respectively. Mappings of Σ^ω into S are called *ω-series* and their collection is denoted by $S\langle\langle\Sigma^\omega\rangle\rangle$. See [20, 16] for more information. Examples of series in $S\langle\Sigma^*\rangle$ for a semiring $\langle S, +, \cdot, 0, 1\rangle$ are 0, w, sw for $s \in S$ and $w \in \Sigma^*$, defined by

$$\begin{aligned}(0, w) &= 0 \text{ for all } w,\\ (w, w) &= 1 \text{ and } (w, w') = 0 \text{ for } w \neq w',\\ (sw, w) &= s \text{ and } (sw, w') = 0 \text{ for } w \neq w'.\end{aligned}$$

Consider a starsemiring-omegasemimodule pair (A, V). Following Bloom, Ésik [2], we define a matrix operation $^\omega : A^{n\times n} \to V^{n\times 1}$ on a starsemiring-omegasemimodule pair (A, V) as follows. If $n = 0$, M^ω is the unique element of V^0, and if $n = 1$, so that $M = (a)$, for some $a \in A$, $M^\omega = (a^\omega)$. Assume now that $n > 1$ and write M as

$$M = \begin{pmatrix} a & b\\ c & d\end{pmatrix}, \tag{1}$$

where a, b, c and d are submatrices of M, called *blocks* of M. Then

$$M^\omega = \begin{pmatrix}(a + bd^*c)^\omega + (a + bd^*c)^*bd^\omega\\ (d + ca^*b)^\omega + (d + ca^*b)^*ca^\omega\end{pmatrix}.$$

Following Ésik, Kuich [15], we define matrix operations ${}^{\omega,k}\colon A^{n\times n}\to V^{n\times 1}$ for $0\le k\le n$ as follows. Assume that $M\in A^{n\times n}$ is decomposed into blocks a,b,c,d as in (1), but with a of dimension $k\times k$ and d of dimension $(n-k)\times(n-k)$. Then

$$M^{\omega,k}=\begin{pmatrix}(a+bd^*c)^\omega\\ d^*c(a+bd^*c)^\omega\end{pmatrix}.$$

Observe that $M^{\omega,0}=0$ and $M^{\omega,n}=M^\omega$. Intuitively, M can be interpreted as an adjacency matrix and $M^{\omega,k}$ are infinite paths where the first k states are repeated states, i.e., states that are Büchi-accepting.

▶ **Example 1.** Formal languages are covered by our model. Let $\langle\mathbb{B},\vee,\wedge,0,1\rangle$ be the Boolean semiring. Then let $0^*=1^*=1$ and take infima as infinite products. This makes $\mathbb{B}$ a continuous star-omega and commutative semiring. It then follows that $\mathbb{B}\langle\langle\Sigma^*\rangle\rangle\times\mathbb{B}\langle\langle\Sigma^\omega\rangle\rangle$ is isomorphic to formal languages of finite and infinite words with the usual operations.

The semiring $\langle\mathbb{N}^\infty,+,\cdot,0,1\rangle$ with $\mathbb{N}^\infty=\mathbb{N}\cup\{\infty\}$ and the natural infinite product operation of numbers is a continuous star-omega and commutative semiring.

The tropical semiring $\langle\mathbb{N}^\infty,\min,+,\infty,0\rangle$ with the usual infinite sum operation as infinite product is a commutative semiring and a continuous star-omega semiring.

3 Mixed ω-Algebraic Systems

This and the next section describe the Greibach normal form for mixed ω-algebraic systems.

Throughout this paper, *S is a continuous, and therefore complete, star-omega semiring with the underlying semiring S being commutative*; and Σ denotes an alphabet.

By Theorem 5.5.5 of Ésik, Kuich [16], $(S\langle\langle\Sigma^*\rangle\rangle,S\langle\langle\Sigma^\omega\rangle\rangle)$ is a complete semiring-semimodule pair, hence a Conway semiring-semimodule pair, satisfying $\epsilon^\omega=0$ (for Conway semiring-semimodule pairs, cf. Ésik, Kuich [16], page 106). Hence, $S\langle\langle\Sigma^*\rangle\rangle\times S\langle\langle\Sigma^\omega\rangle\rangle$ is a generalized starquemiring.

In the sequel, x and z denote vectors of dimension n and m, respectively, i.e., $x=(x_1,\dots,x_n)$, $z=(z_1,\dots,z_m)$. It will be clear from the context whether they are used as row or as column vectors. Similar conventions hold for vectors p, σ and τ. Moreover, X denotes the set of variables $\{x_1,\dots,x_n\}$ for $S\langle\langle\Sigma^*\rangle\rangle$, while $\{z_1,\dots,z_m\}$ is the set of variables for $S\langle\langle\Sigma^\omega\rangle\rangle$.

A *mixed ω-algebraic system over the quemiring $S\langle\langle\Sigma^*\rangle\rangle\times S\langle\langle\Sigma^\omega\rangle\rangle$* consists of an algebraic system over $S\langle\langle\Sigma^*\rangle\rangle$

$$x=p(x),\qquad p\in(S\langle(\Sigma\cup X)^*\rangle)^{n\times 1}$$

and a linear system over $S\langle\langle\Sigma^\omega\rangle\rangle$

$$z=\varrho(x)z,\qquad \varrho\in(S\langle(\Sigma\cup X)^*\rangle)^{m\times m}.$$

The pair $(\sigma,\tau)\in(S\langle\langle\Sigma^*\rangle\rangle)^n\times(S\langle\langle\Sigma^\omega\rangle\rangle)^m$ is a *solution* of the mixed ω-algebraic system

$$x=p(x),\quad z=\varrho(x)z\,,\qquad\text{if}\qquad \sigma=p(\sigma),\quad \tau=\varrho(\sigma)\tau\,.$$

Observe that, by Theorem 5.5.7 of Ésik, Kuich [16], $\tau_k=\varrho(\sigma)^{\omega,k}$ for each $1\le k\le m$ is a solution for the linear system $z=\varrho(\sigma)z$.

A solution $(\sigma_1,\dots,\sigma_n)$ of the algebraic system $x=p(x)$ is termed *least solution* if

$$\sigma_i\le\tau_i,\qquad\text{for each } 1\le i\le n,$$

for all solutions $(\tau_1,\dots,\tau_n)$ of $x=p(x)$.

If σ is the least solution of $x = p(x)$, then $z = \varrho(\sigma)z$ is called an $S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle$-linear system and $(\sigma, \tau_k) = (\sigma, \varrho(\sigma)^{\omega,k})$, where $k \in \{0, 1, \ldots, m\}$, is called k^{th}-*canonical solution* of $x = p(x)$, $z = \varrho(x)z$. Each k^{th}-canonical solution is also called a *canonical solution*.

Recall that $S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle$ comprises the components of least solutions of algebraic systems

$$x_i = p_i, \quad (1 \le i \le n) \qquad \text{where } p_i \in S\langle(\Sigma \cup X)^*\rangle \text{ for } 1 \le i \le n\,.$$

We define $S^{\mathrm{alg}}\langle\langle\Sigma^\omega\rangle\rangle$ to be the collection of all components of vectors $M^{\omega,k}$, where $M \in (S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle)^{n\times n}$, $n \ge 1$, and $k \in \{1, \ldots, n\}$. Moreover, ω-$\mathfrak{Rat}(S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle)$ is defined to be the ω-Kleene closure of (i.e., the generalized starquemiring generated by) $S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle$.

▶ **Example 2.** We consider the following mixed ω-algebraic system over the quemiring $\mathbb{N}^\infty\langle\langle\Sigma^*\rangle\rangle \times \mathbb{N}^\infty\langle\langle\Sigma^\omega\rangle\rangle$ for the tropical semiring $\langle\mathbb{N}^\infty, \min, +, \infty, 0\rangle$

$$\begin{aligned} x_1 &= 1ax_1b + 1ab & z_1 &= cz_1 \\ & & z_2 &= x_1z_1 + z_1 \end{aligned}$$

where $a, b, c \in \Sigma$ and using the natural number 1.

Then for the algebraic system $x = p(x)$ over $\mathbb{N}^\infty\langle\langle\Sigma^*\rangle\rangle$, we get the least solution $\sigma = a^nb^n \mapsto n$. The first canonical solution of the mixed ω-algebraic system $x = p(x)$, $z = \varrho(x)z$ over $\mathbb{N}^\infty\langle\langle\Sigma^*\rangle\rangle \times \mathbb{N}^\infty\langle\langle\Sigma^\omega\rangle\rangle$ is then $(\sigma, c^\omega \mapsto 0, a^nb^nc^\omega \mapsto n)$. Hence the series $a^nb^nc^\omega \mapsto n$ is ω-algebraic but it is clearly not recognizable by a weighted automaton without stack.

Now we have the following characterization of algebraic and ω-algebraic series.

▶ **Theorem 3.** *Let S be a continuous complete star-omega semiring with the underlying semiring S being commutative and let Σ be an alphabet. Then the following statements are equivalent for $(s, v) \in S\langle\langle\Sigma^*\rangle\rangle \times S\langle\langle\Sigma^\omega\rangle\rangle$:*

(i) $(s, v) \in S^{alg}\langle\langle\Sigma^*\rangle\rangle \times S^{alg}\langle\langle\Sigma^\omega\rangle\rangle$,

(ii) $(s, v) \in \omega\text{-}\mathfrak{Rat}(S^{alg}\langle\langle\Sigma^*\rangle\rangle)$,

(iii) $(s, v) = \|\mathfrak{A}\|$, *where $\mathfrak{A}$ is a finite $S^{alg}\langle\langle\Sigma^*\rangle\rangle$-automaton over $S\langle\langle\Sigma^*\rangle\rangle \times S\langle\langle\Sigma^\omega\rangle\rangle$,*

(iv) $s \in S^{alg}\langle\langle\Sigma^*\rangle\rangle$ *and* $v = \sum_{1\le j\le l} s_jt_j^\omega$ *for some $l \ge 0$, where $s_j, t_j \in S^{alg}\langle\langle\Sigma^*\rangle\rangle$,*

(v) *(s, v) is component of the automata-theoretic solution of an $S^{alg}\langle\langle\Sigma^*\rangle\rangle$-linear system over $S\langle\langle\Sigma^*\rangle\rangle \times S\langle\langle\Sigma^\omega\rangle\rangle$,*

(vi) *(s, v) is component of the canonical solution of a mixed ω-algebraic system over $S\langle\langle\Sigma^*\rangle\rangle \times S\langle\langle\Sigma^\omega\rangle\rangle$.*

Proof. The statements (ii), (iii) and (iv) are equivalent by Theorem 5.4.9 (see also Theorem 5.6.6) of Ésik, Kuich [16]. ◀

4 Greibach Normal Form for Mixed ω-Algebraic Systems

In this section we show that for any element of $S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle \times S^{\mathrm{alg}}\langle\langle\Sigma^\omega\rangle\rangle$ there exists a mixed ω-algebraic system in Greibach normal form such that this element is a component of a solution of this mixed ω-algebraic system. Similar to the definition for algebraic systems on finite words (cf. also Greibach [18]), a mixed ω-algebraic system

$$x = p(x), \quad z = \varrho(x)z$$

is in *Greibach normal form* if

$$\begin{aligned} &\mathrm{supp}(p_i(x)) \subseteq \{\epsilon\} \cup \Sigma \cup \Sigma X \cup \Sigma XX, && \text{for all } 1 \le i \le n, \quad \text{and} \\ &\mathrm{supp}(\varrho_{ij}(x)) \subseteq \Sigma \cup \Sigma X, && \text{for all } 1 \le i, j \le m\,. \end{aligned}$$

For the construction of the Greibach normal form we need a corollary to Theorem 3.

▶ **Corollary 4.** *The following statement for $(s, \upsilon) \in S\langle\langle\Sigma^*\rangle\rangle \times S\langle\langle\Sigma^\omega\rangle\rangle$ is equivalent to the statements* (i) *to* (vi) *of Theorem 3:*
$s \in S^{alg}\langle\langle\Sigma^*\rangle\rangle$ and $\upsilon = \sum_{1\le j\le l} s_j t_j^\omega$ for some $l \ge 0$, where $s_j, t_j \in S^{alg}\langle\langle\Sigma^*\rangle\rangle$ with $(t_j, \epsilon) = 0$; moreover $(s_j, \epsilon) = 0$ or $s_j = (s_j, \epsilon)\epsilon$.

Proof. The proof is an easy case distinction. ◀

We now assume that $(s, \upsilon) \in S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle \times S^{\mathrm{alg}}\langle\langle\Sigma^\omega\rangle\rangle$ is given in the form of Corollary 4 with $l = 1$. By Theorem 2.4.10 of Ésik, Kuich [16], there exist algebraic systems in Greibach normal form whose first component of their least solutions equals s_1, t_1.

Firstly, we deal with the case $(s_1, \epsilon) = 0$. Let

$$x_i = p_i(x) + \sum_{1\le j\le n} p_{ij}(x)x_j, \quad \text{for each } 1 \le i \le n, \tag{$*$}$$

where $\mathrm{supp}(p_i(x)) \subseteq \Sigma \cup \Sigma X$, $\mathrm{supp}(p_{ij}(x)) \subseteq \Sigma X$, be the algebraic system in Greibach normal form for s_1 and

$$x'_i = p'_i(x') + \sum_{1\le j\le m} p'_{ij}(x')x'_j, \quad \text{for each } 1 \le i \le m, \tag{$**$}$$

where $\mathrm{supp}(p'_i(x')) \subseteq \Sigma \cup \Sigma X'$, $\mathrm{supp}(p_{ij}(x')) \subseteq \Sigma X'$, be the algebraic system in Greibach normal form for t_1. Let σ and σ' with $\sigma_1 = s_1$ and $\sigma'_1 = t_1$ be the least solutions of $(*)$ and $(**)$, respectively.

Consider now the mixed ω-algebraic system consisting of the algebraic system $(*)$, $(**)$ over $S\langle\langle\Sigma^*\rangle\rangle$ and the linear system over $S\langle\langle\Sigma^\omega\rangle\rangle$

$$\begin{aligned} z'' &= p'_1(x')z'' + \sum_{1\le j\le m} p'_{1j}(x')z'_j\,, \\ z'_i &= p'_i(x')z'' + \sum_{1\le j\le m} p'_{ij}(x')z'_j, \qquad \text{for } 1 \le i \le m\,, \\ z_i &= p_i(x)z'' + \sum_{1\le j\le n} p_{ij}(x)z_j, \qquad \text{for } 1 \le i \le n\,. \end{aligned} \tag{$***$}$$

Observe that the mixed ω-algebraic system is in Greibach normal form. We then order the variables of the mixed ω-algebraic system $(*)$, $(**)$, $(***)$ as $x_1, \ldots, x_n; x'_1, \ldots, x'_m; z''$; $z'_1, \ldots, z'_m; z_1, \ldots, z_n$. Observe that $\sigma'_1\sigma'^\omega_1 = \sigma'^\omega_1$.

The next lemma states that the system $(*)$, $(**)$, $(***)$ is the mixed ω-algebraic system in Greibach normal whose canonical solution indeed contains a component $\sigma_1\sigma'^\omega_1 = s_1t_1^\omega$ as described in the statement of Corollary 4.

▶ **Lemma 5.** *The solution*

$$(\sigma_1, \ldots, \sigma_n; \sigma'_1, \ldots, \sigma'_m; \sigma'_1\sigma'^\omega_1; \sigma'_1\sigma'^\omega_1, \ldots, \sigma'_m\sigma'^\omega_1; \sigma_1\sigma'^\omega_1, \ldots, \sigma_n\sigma'^\omega_1) \tag{2}$$

is the first canonical solution of the mixed ω-algebraic system $(*)$, $(**)$, $(***)$.

Secondly, we deal with the case $s_1 = (s_1, \epsilon)\epsilon$. Consider now the mixed ω-algebraic system consisting of $(**)$ and the linear system over $S\langle\langle\Sigma^\omega\rangle\rangle$

$$\begin{aligned} z'' &= p'_1(x')z'' + \sum_{1\le j\le m} p'_{1j}(x')z'_j\,, \\ z'_i &= p'_i(x')z'' + \sum_{1\le j\le m} p'_{ij}(x')z'_j,\ 1 \le i \le m\,, \\ z_1 &= (s_1, \epsilon)p'_1(x')z'' + (s_1, \epsilon) \sum_{1\le j\le m} p'_{1j}(x')z'_j\,. \end{aligned} \tag{$****$}$$

▶ **Lemma 6.** *The solution*

$$(\sigma'_1, \ldots, \sigma'_m; \sigma'_1\sigma_1'^{\omega}; \sigma'_1\sigma_1'^{\omega}, \ldots, \sigma'_m\sigma_1'^{\omega}; (s_1, \epsilon)\sigma_1'^{\omega}) \,. \tag{3}$$

is the first canonical solution of the mixed ω-algebraic system (∗∗), (∗∗∗∗).

We now consider general sums of series of the above form. The next lemma shows how to construct a mixed ω-algebraic system whose canonical solution is the sum of the canonical solutions of multiple mixed ω-algebraic systems as given in the Lemmas 5 and 6.

▶ **Lemma 7.** *Let $(s, \upsilon) \in S^{alg}\langle\langle\Sigma^*\rangle\rangle \times S^{alg}\langle\langle\Sigma^\omega\rangle\rangle$ be given in the form of Corollary 4. Then there exists a mixed ω-algebraic system in Greibach normal form such that υ is a component of its l-th canonical solution.*

Our first main result is the following.

▶ **Theorem 8.** *The following statement for $(s, \upsilon) \in S\langle\langle\Sigma^*\rangle\rangle \times S\langle\langle\Sigma^\omega\rangle\rangle$ is equivalent to the statements of Theorem 3:*
(s, υ) is component of a canonical solution of a mixed ω-algebraic system over $S\langle\langle\Sigma^*\rangle\rangle \times S\langle\langle\Sigma^\omega\rangle\rangle$ in Greibach normal form.

Proof. The above statement trivially implies statement (vi) of Theorem 3. By Corollary 4 and Lemma 7, the statements of Theorem 3 imply the above statement. ◀

5 Simple Reset Pushdown Automata

In this second part of the paper, we want to show that weighted ω-pushdown automata can be transformed into a simple form. The next section will prove this result for ω-algebraic series that are a component of a solution of an ω-algebraic system in Greibach normal form. For the proof, we will need the corresponding result for finite words as an intermediate step. This result, the expressive equivalence of algebraic series (of finite words) and (weighted) simple reset pushdown automata, has been established in [7]. We recall the construction of the weighted simple reset pushdown automata here for the convenience of the reader, as variants of these automata will be used in Section 6 for ω-algebraic series.

Following Kuich, Salomaa [20] and Kuich [19], we introduce pushdown transitions matrices. These matrices can be considered as adjacency matrices of graphs representing automata. A special form, the reset pushdown matrices, is used for pushdown automata starting with an empty stack and allowing the automaton to push onto the empty stack. Here, we are interested in simple reset pushdown matrices, introduced in [7]. This simple form allows the automaton only to push one symbol, to pop one symbol or to ignore the stack. The corresponding automata, the simple reset pushdown automata are a generalization of the unweighted automata used in [6]. They do not use ϵ-transitions and don't allow the inspection of the topmost stack symbol.

Let Γ be an alphabet, called *pushdown alphabet* and let $n \geq 1$. A matrix $\bar{M} \in (S^{n\times n})^{\Gamma^*\times\Gamma^*}$ is called a *pushdown matrix* (with *pushdown alphabet* Γ and *state set* $\{1, \ldots, n\}$) if

(i) for each $p \in \Gamma$ there exist only finitely many blocks $\bar{M}_{p,\pi}$, $\pi \in \Gamma^*$, that are unequal to 0;
(ii) for all $\pi_1, \pi_2 \in \Gamma^*$,

$$\bar{M}_{\pi_1,\pi_2} = \begin{cases} \bar{M}_{p,\pi}, & \text{if there exist } p \in \Gamma, \pi, \pi' \in \Gamma^* \text{ with } \pi_1 = p\pi' \text{ and } \pi_2 = \pi\pi', \\ 0, & \text{otherwise.} \end{cases}$$

Intuitively, the infinite pushdown matrix $\bar{M}$ is (ii) fully represented only by the blocks $\bar{M}_{p,\pi}$ where $p \in \Gamma$, $\pi \in \Gamma^*$ and (i) only finitely many such blocks are nonzero.

A matrix $M \in (S^{n\times n})^{\Gamma^*\times\Gamma^*}$ is called *row-finite* if $\{\pi' \mid M_{\pi,\pi'} \neq 0\}$ is finite for all $\pi \in \Gamma^*$. Let Γ be a pushdown alphabet and $\{1,\dots,n\}$, $n \geq 1$, be a set of states. A *reset matrix* $M_R \in (S^{n\times n})^{\Gamma^*\times\Gamma^*}$ is a row-finite matrix such that

$$(M_R)_{\pi_1,\pi_2} = 0 \qquad \text{for } \pi_1,\pi_2 \in \Gamma^* \text{ with } \pi_1 \neq \epsilon\,.$$

A *reset pushdown matrix* $M \in (S^{n\times n})^{\Gamma^*\times\Gamma^*}$ is the sum $M = M_R + \bar{M}$ of a reset matrix M_R and a pushdown matrix $\bar{M}$.

Intuitively, a reset pushdown matrix is similar to a pushdown matrix with the additional possibility to push onto the empty stack, i.e., $M_{\epsilon,\pi}$ is allowed to be nonzero. Note that reset pushdown matrices are still finitely represented because of the row-finiteness.

A reset pushdown matrix M is called *simple* if $M \in \big((S\langle\Sigma\rangle)^{n\times n}\big)^{\Gamma^*\times\Gamma^*}$ for some $n \geq 1$, and for all $p, p_1 \in \Gamma$,

$$M_{p,\epsilon},\ M_{p,p} = M_{\epsilon,\epsilon} \text{ and } M_{p,p_1p} = M_{\epsilon,p_1},$$

are the only blocks $M_{\pi,\pi'}$, where $\pi \in \{\epsilon, p\}$ and $\pi' \in \Gamma^*$, that may be unequal to the zero matrix 0.

Hence, a simple reset pushdown matrix M is defined by its blocks $M_{\epsilon,\epsilon}$ and $M_{p,\epsilon}$, $M_{\epsilon,p}$ ($p \in \Gamma$). Intuitively, the automata will only be allowed to ignore the stack (modeled by $M_{\epsilon,\epsilon}$), pop one symbol ($M_{p,\epsilon}$) or push one symbol ($M_{\epsilon,p}$). Note also that the matrix $M \in ((S\langle\Sigma\rangle)^{n\times n})^{\Gamma^*\times\Gamma^*}$ forbids ϵ-transitions. Moreover, the equalities $M_{p,p} = M_{\epsilon,\epsilon}$ and $M_{p,p_1p} = M_{\epsilon,p_1}$ imply that the next transition does not depend on the topmost symbol of the stack except when popping it (modeled by $M_{p,\epsilon}$). An example of a simple reset pushdown matrix can be found in Example 14.

A *reset pushdown automaton* (with input alphabet Σ) $\mathfrak{A} = (n, \Gamma, I, M, P)$ is given by

- a *set of states* $\{1,\dots,n\}$, $n \geq 1$,
- a *pushdown alphabet* Γ,
- a reset pushdown matrix $M \in ((S\langle\Sigma \cup \{\epsilon\}\rangle)^{n\times n})^{\Gamma^*\times\Gamma^*}$ called *transition matrix*,
- a row vector $I \in (S\langle\{\epsilon\}\rangle)^{1\times n}$, called *initial state vector*,
- a column vector $P \in (S\langle\{\epsilon\}\rangle)^{n\times 1}$, called *final state vector*.

The *behavior* $\|\mathfrak{A}\|$ of a reset pushdown automaton $\mathfrak{A}$ is defined by

$$\|\mathfrak{A}\| = I(M^*)_{\epsilon,\epsilon}P\,.$$

A reset pushdown automaton $\mathfrak{A} = (n, \Gamma, I, M, P)$ is called *simple* if M is a simple reset pushdown matrix. Example 14 shows a simple ω-reset pushdown automaton.

Given a series $r \in S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle$, we want to construct a simple reset pushdown automaton with behavior r. By Theorems 5.10 and 5.4 of [19], r is a component of the unique solution of a strict algebraic system in Greibach normal form.

We only consider the algebraic series r with $(r,\epsilon) = 0$; cf. [7] for the other case. So we assume without loss of generality that r is the x_1-component of the unique solution of the algebraic system (4) with variables $x_1,\dots,x_n$

$$x_i = p_i,\ \ 1 \leq i \leq n,$$

of the form

$$x_i = \sum_{1\leq j,k\leq n}\sum_{a\in\Sigma}(p_i, ax_jx_k)ax_jx_k + \sum_{1\leq j\leq n}\sum_{a\in\Sigma}(p_i, ax_j)ax_j + \sum_{a\in\Sigma}(p_i,a)a\,. \tag{4}$$

As shown in [7], we can construct the simple reset pushdown automaton $\mathfrak{A}_s = (n+1, \Gamma, I_s, M, P)$, $1 \leq s \leq n$, with $r = \|\mathfrak{A}_1\|$ as follows:
We let $\Gamma = \{x_1, \ldots, x_n\}$; we also denote the state $n+1$ by f; the entries of M of the form $(M_{x_k,x_k})_{i,j}$, $(M_{x_k,\epsilon})_{i,j}$, $(M_{\epsilon,x_k})_{i,j}$, $(M_{\epsilon,\epsilon})_{i,j}$, $(M_{\epsilon,\epsilon})_{i,f}$, where $1 \leq i, j, k \leq n$, that may be unequal to 0 are

$$\begin{aligned}(M_{\epsilon,x_k})_{i,j} &= \sum_{a\in\Sigma}(p_i, ax_jx_k)a\,,\\ (M_{x_k,x_k})_{i,j} = (M_{\epsilon,\epsilon})_{i,j} &= \sum_{a\in\Sigma}(p_i, ax_j)a\,,\\ (M_{x_k,\epsilon})_{i,k} = (M_{x_k,x_k})_{i,f} = (M_{\epsilon,\epsilon})_{i,f} &= \sum_{a\in\Sigma}(p_i, a)a\,;\end{aligned}$$

we further put $(I_s)_s = \epsilon$, $(I_s)_i = 0$ for $1 \leq i \leq s-1$ and $s+1 \leq i \leq n+1$; finally let $P_f = \epsilon$ and $P_j = 0$ for $1 \leq j \leq n$;

The following motivation will be essential for our later construction for ω-pushdown automata. Intuitively, the variables in the algebraic system are simulated by states in the simple reset pushdown automaton $\mathfrak{A}_s$. By the Greibach normal form, only two variables on the right-hand side are allowed. The first is modeled directly by changing the state, the second is pushed to the pushdown tape and the state is changed to it later when the variable is popped again. The special final state f will only be used as the last state.

Note that $(M_{x_k,x_k})_{i,f}$ allows the automaton to change to the final state with a non-empty pushdown tape. This is an artificial addition to fit the definition of simple reset pushdown matrices. If the simple reset automaton is not popping a symbol from the pushdown tape, it cannot distinguish between different pushdown states. Even though the automaton can enter the final state too early, it can not continue from there as it is a sink.

Observe that $\|\mathfrak{A}_s\| = ((M^*)_{\epsilon,\epsilon})_{s,f}$ for all $1 \leq s \leq n$.

This simple reset pushdown matrix M is called the simple pushdown matrix *induced* by the Greibach normal form (4). The simple reset pushdown automata $\mathfrak{A}_s$, $1 \leq s \leq n$, are called the simple reset pushdown automata *induced* by the Greibach normal form (4).

The following (main) theorem of [7] states that the behavior of the simple reset pushdown automata induced by the Greibach normal form (4) is the unique solution of the original algebraic system (4).

▶ **Theorem 9** (Theorem 11 of [7]). *The unique solution of the algebraic system* (4) *is*

$$(\|\mathfrak{A}_1\|, \ldots, \|\mathfrak{A}_n\|) = (((M^*)_{\epsilon,\epsilon})_{1,f}, \ldots, ((M^*)_{\epsilon,\epsilon})_{n,f})\,.$$

▶ **Corollary 10** (Corollary 12 of [7]). *Let $r \in S^{alg}\langle\langle\Sigma^*\rangle\rangle$. Then there exists a simple reset pushdown automaton with behavior r.*

6 Simple ω-Reset Pushdown Automata

This section will prove that simple ω-reset pushdown automata can be obtained from ω-algebraic systems in Greibach normal form. We first prove some results for infinite applications of simple reset pushdown matrices. Then we introduce simple ω-reset pushdown automata and the main theorem will show that they can recognize all ω-algebraic series that are solutions of ω-algebraic systems in Greibach normal form.

In the sequel, (S, V) *is a complete semiring-semimodule pair.*

We will use sets P_l comprising infinite sequences over $\{1, \dots, n\}$ as defined in [8]:

$$P_l = \{(j_1, j_2, \dots) \in \{1, \dots, n\}^\omega \mid j_t \leq l \text{ for infinitely many } t \geq 1\}\,.$$

Observe the following summation identity: Assume that $A_1, A_2, \dots$ are matrices in $S^{n \times n}$. Then for $0 \leq l \leq n$, $1 \leq j \leq n$, and $m \geq 1$, we have

$$\sum_{(j_1,j_2,\dots)\in P_l} (A_1)_{j,j_1}(A_2)_{j_1,j_2} \cdots = \sum_{1 \leq j_1,\dots,j_m \leq n} (A_1)_{j,j_1} \cdots (A_m)_{j_{m-1},j_m} \sum_{(j_{m+1},j_{m+2},\dots)\in P_l} (A_{m+1})_{j_m,j_{m+1}} \cdots \,.$$

By Theorem 5.5.1 of Ésik, Kuich [16] we obtain, for a finite matrix A and for $0 \leq l \leq n$, the equality $AA^{\omega,l} = A^{\omega,l}$. By Theorem 6 of Droste, Ésik, Kuich [8], we have a similar result for pushdown matrices. We will now show the same equality for a reset pushdown matrix M.

▶ **Theorem 11.** *Let (S, V) be a complete semiring-semimodule pair and $M \in (S^{n\times n})^{\Gamma^* \times \Gamma^*}$ be a reset pushdown matrix. Then*

(i) $M^{\omega,l} = MM^{\omega,l}$, *for each* $0 \leq l \leq n$,

(ii) $(M^\omega)_p = (\bar{M}^\omega)_p + (\bar{M}^*)_{p,\epsilon}(M^\omega)_\epsilon$, *for any* $p \in \Gamma$,

(iii) $(M^{\omega,l})_p = (\bar{M}^{\omega,l})_p + (\bar{M}^*)_{p,\epsilon}(M^{\omega,l})_\epsilon$, *for each* $0 \leq l \leq n$ *and* $p \in \Gamma$.

Proof.

(i) The proof is similar to the proof of Theorem 6 of [8] but we also need to handle empty pushdown tapes.

(ii) We obtain, for $p \in \Gamma$,

$$\begin{aligned}(M^\omega)_p &= \sum_{\pi_1,\pi_2,\dots\in\Gamma^+} M_{p,\pi_1}M_{\pi_1,\pi_2}\cdots + \sum_{t\geq 1}\sum_{\pi_1,\dots,\pi_{t-1}\in\Gamma^+} M_{p,\pi_1}\cdots M_{\pi_{t-1},\epsilon}(M^\omega)_\epsilon \\ &= (\bar{M}^\omega)_p + \sum_{t\geq 1}(\bar{M}^t)_{p,\epsilon}(M^\omega)_\epsilon \\ &= (\bar{M}^\omega)_p + (\bar{M}^*)_{p,\epsilon}(M^\omega)_\epsilon\,.\end{aligned}$$

(iii) The proof is similar to the proof of (ii) but more technical as it needs to consider the repeated states. ◀

▶ **Lemma 12.** *Let (S, V) be a complete semiring-semimodule pair. Let M be a simple reset pushdown matrix. Then,*

(i) $(M^\omega)_p = (M^\omega)_\epsilon + (M^*)_{\epsilon,\epsilon}M_{p,\epsilon}(M^\omega)_\epsilon$ *for* $p \in \Gamma$,

(ii) $(M^{\omega,l})_p = (M^{\omega,l})_\epsilon + (M^*)_{\epsilon,\epsilon}(M_{p,\epsilon})(M^{\omega,l})_\epsilon$ *for each* $0 \leq l \leq n$ *and* $p \in \Gamma$.

Proof. Only (i): We obtain, for $p \in \Gamma$,

$$\begin{aligned}(M^\omega)_p &= \sum_{\pi_1,\pi_2,\dots\in\Gamma^*} M_{p,\pi_1}M_{\pi_1,\pi_2}\cdots \\ &= \sum_{\pi_1,\pi_2,\dots\in\Gamma^*} M_{p,\pi_1 p}M_{\pi_1 p,\pi_2 p}\cdots + \sum_{t\geq 0}\sum_{\pi_1,\dots,\pi_{t-1}\in\Gamma^*} M_{p,\pi_1 p}\cdots M_{\pi_{t-1}p,p}M_{p,\epsilon}(M^\omega)_\epsilon \\ &= (M^\omega)_\epsilon + \big(\sum_{t\geq 0}\sum_{\pi_1,\dots,\pi_{t-1}\in\Gamma^*} M_{\epsilon,\pi_1}\cdots M_{\pi_{t-1},\epsilon}\big)M_{p,\epsilon}(M^\omega)_\epsilon \\ &= (M^\omega)_\epsilon + \sum_{t\geq 0}(M^t)_{\epsilon,\epsilon}M_{p,\epsilon}(M^\omega)_\epsilon \\ &= (M^\omega)_\epsilon + (M^*)_{\epsilon,\epsilon}M_{p,\epsilon}(M^\omega)_\epsilon\,.\end{aligned}$$

◀

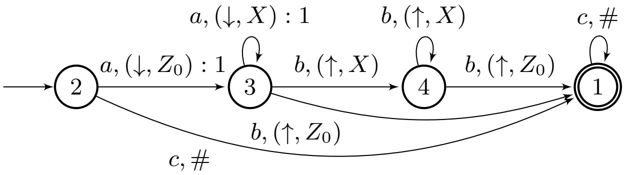


Figure 1 Example 14: Weighted simple ω-pushdown automaton, where $(\downarrow, X)$ means push symbol X, $(\uparrow, X)$ means pop X, and $\#$ leaves the stack unaltered. All shown transitions have a weight equal to the natural number 0 except the two transitions reading letter a and pushing a symbol onto the stack that have weight 1. All other possible transitions have weight ∞.

▶ **Lemma 13.** *Let M be induced by the Greibach normal form (4). Then, for all $1 \leq j, k \leq n$ and $0 \leq l \leq n$,*

$$((M^{\omega,l})_{x_k})_j = ((M^{\omega,l})_\epsilon)_j + ((M^*)_{\epsilon,\epsilon})_{j,f}((M^{\omega,l})_\epsilon)_k \, .$$

Proof. By Lemma 12(ii), we have

$$\begin{aligned} ((M^{\omega,l})_{x_k})_j &= \left[(M^{\omega,l})_\epsilon + (M^*)_{\epsilon,\epsilon}(M_{x_k,\epsilon})(M^{\omega,l})_\epsilon\right]_j \\ &= ((M^{\omega,l})_\epsilon)_j + \left[(M^*)_{\epsilon,\epsilon}(M_{x_k,\epsilon})(M^{\omega,l})_\epsilon\right]_j \end{aligned}$$

Then for $1 \leq j, k \leq n$, we have

$$\begin{aligned} \left((M^*)_{\epsilon,\epsilon}(M_{x_k,\epsilon})(M^{\omega,l})_\epsilon\right)_j &= \sum_{1 \leq t_1, t_2 \leq f} ((M^*)_{\epsilon,\epsilon})_{j,t_1}(M_{x_k,\epsilon})_{t_1,t_2}((M^{\omega,l})_\epsilon)_{t_2} \\ &= \sum_{1 \leq t_1 \leq f} ((M^*)_{\epsilon,\epsilon})_{j,t_1}(M_{\epsilon,\epsilon})_{t_1,f}((M^{\omega,l})_\epsilon)_k \\ &= ((M^*)_{\epsilon,\epsilon}M_{\epsilon,\epsilon})_{j,f}((M^{\omega,l})_\epsilon)_k \\ &= ((M^*)_{\epsilon,\epsilon})_{j,f}((M^{\omega,l})_\epsilon)_k \, . \end{aligned}$$

The second equality holds because we defined $(M_{x_k,\epsilon})_{t_1,t_2} = 0$ for $t_2 \neq k$ and $(M_{x_k,\epsilon})_{t_1,k} = (M_{\epsilon,\epsilon})_{t_1,f}$ for induced simple pushdown matrices. The result follows. ◀

Next, an ω-reset pushdown automaton

$$\mathfrak{A} = (n, \Gamma, I, M, P, l)$$

is given by a reset pushdown automaton (n, Γ, I, M, P) and an integer l with $0 \leq l \leq n$, which indicates that $1, \ldots, l$ are the repeated states of $\mathfrak{A}$. The behavior $\|\mathfrak{A}\|$ of this ω-reset pushdown automaton $\mathfrak{A}$ is defined by

$$\|\mathfrak{A}\| = I(M^*)_{\epsilon,\epsilon}P + I(M^{\omega,l})_\epsilon \, .$$

The ω-reset pushdown automaton $\mathfrak{A} = (n, \Gamma, I, M, P, l)$ is called *simple* if M is a simple reset pushdown matrix.

▶ **Example 14.** Figure 1 shows a simple ω-reset pushdown automaton $\mathcal{A} = (4, \Gamma, I, M, P, 1)$ over the quemiring $\mathbb{N}^\infty\langle\langle\Sigma^*\rangle\rangle \times \mathbb{N}^\infty\langle\langle\Sigma^\omega\rangle\rangle$ for the tropical semiring $\langle\mathbb{N}^\infty, \min, +, \mathbb{0} = \infty, \mathbb{1} = 0\rangle$ with $\Sigma = \{a, b, c\}$, $\Gamma = \{Z_0, X\}$, $I_2 = 0$, $I_i = \infty$ for $i \neq 2$ and $P_i = \infty$ for all $1 \leq i \leq 4$. The adjacency matrix M of the automaton is a simple reset pushdown matrix. As an indication, M is defined with $(M_{\epsilon,\epsilon})_{1,1} = (M_{\epsilon,\epsilon})_{2,1} = 0c$, $(M_{\epsilon,Z_0})_{2,3} = 1a$, etc., resulting in e.g.,

$$M_{\epsilon,\epsilon} = \begin{pmatrix} 0c & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ 0c & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \end{pmatrix} \text{ and finally } M = \begin{pmatrix} M_{\epsilon,\epsilon} & M_{\epsilon,Z_0} & M_{\epsilon,X} & \cdots \\ M_{Z_0,\epsilon} & M_{\epsilon,\epsilon} & \mathbb{0} & \cdots \\ M_{X,\epsilon} & \mathbb{0} & M_{\epsilon,\epsilon} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix},$$

where the excluded part of M can be derived from the rules of pushdown and simple reset pushdown matrices. The automaton $\mathcal{A}$ has the behavior $a^n b^n c^\omega \mapsto n$, similar to the mixed ω-algebraic system in Example 2.

Now, for a series $r \in S^{\mathrm{alg}}\langle\langle\Sigma^*\rangle\rangle \times S^{\mathrm{alg}}\langle\langle\Sigma^\omega\rangle\rangle$, we want to construct a simple ω-reset pushdown automaton with behavior r. For our construction, r must be a component of a solution of an ω-algebraic system in Greibach normal form. An ω-algebraic system consists of only one system over the quemiring variables $\{y_1, \ldots, y_n\}$. See [16], pp. 136 for details.

Similar to the definition for mixed ω-algebraic systems, an ω-algebraic system

$$y = p(y)$$

is in *Greibach normal form* if

$$\mathrm{supp}(p_i(y)) \subseteq \{\epsilon\} \cup \Sigma \cup \Sigma Y \cup \Sigma YY, \qquad \text{for all } 1 \leq i \leq n\,.$$

Let r be a component of a solution of the ω-algebraic system (5) in Greibach normal form over the complete semiring-semimodule pair (S, V), i.e., over the quemiring $S \times V$,

$$y_i = \sum_{1\leq j,k\leq n} \sum_{a\in\Sigma} (p_i, ay_jy_k)ay_jy_k + \sum_{1\leq j\leq n} \sum_{a\in\Sigma} (p_i, ay_j)ay_j + \sum_{a\in\Sigma} (p_i, a)a\,. \tag{5}$$

The variables of this system are y_i, $(1 \leq i \leq n)$; they are variables for (S, V). The system (5) induces the following mixed ω-algebraic system:

$$x_i = \sum_{1\leq j,k\leq n} \sum_{a\in\Sigma} (p_i, ay_jy_k)ax_jx_k + \sum_{1\leq j\leq n} \sum_{a\in\Sigma} (p_i, ay_j)ax_j + \sum_{a\in\Sigma} (p_i, a)a\,, \tag{4}$$

and

$$z_i = \sum_{1\leq j,k\leq n} \sum_{a\in\Sigma} (p_i, ay_jy_k)a(z_j + x_jz_k) + \sum_{1\leq j\leq n} \sum_{a\in\Sigma} (p_i, ay_j)az_j\,. \tag{6}$$

Let now, for $1 \leq s \leq n$ and $0 \leq l \leq n$,

$$\mathfrak{A}_s^l = (n+1, \Gamma, I_s, M, P, l)$$

be the simple ω-reset pushdown automata such that $(n+1, \Gamma, I_s, M, P)$, for $1 \leq s \leq n$, are induced by the Greibach normal form (4).

The following theorem states that the induced simple ω-reset pushdown automata behave similar to the solution of system (5). Note that in the semimodule part $(M^{\omega,l})_\epsilon$ of the behavior, state f will never be reached.

► **Theorem 15.** *Let (S, V) be a complete semiring-semimodule pair. Let the simple ω-reset pushdown automata $\|\mathfrak{A}_s^l\|$ for $1 \leq s \leq n$ and $0 \leq l \leq n$ be induced by the Greibach normal form (4). Then, for $0 \leq l \leq n$,*

$$(\|\mathfrak{A}_1^l\|, \ldots, \|\mathfrak{A}_n^l\|) = \big(((M^*)_{\epsilon,\epsilon})_{1,f} + ((M^{\omega,l})_\epsilon)_1, \ldots, ((M^*)_{\epsilon,\epsilon})_{n,f} + ((M^{\omega,l})_\epsilon)_n\big)$$

is a solution of (5).

Proof. By Theorem 9, $(((M^*)_{\epsilon,\epsilon})_{1,f}, \dots, ((M^*)_{\epsilon,\epsilon})_{n,f})$ is a solution of (4). We show that $(((M^{\omega,l})_\epsilon)_1, \dots, ((M^{\omega,l})_\epsilon)_n)$ is a solution of (6) and substitute it into the right sides of (6):

$$\begin{aligned}
&\sum_{1\le j,k\le n}(M_{\epsilon,y_k})_{i,j}\Big(((M^{\omega,l})_\epsilon)_j + ((M^*)_{\epsilon,\epsilon})_{j,f}((M^{\omega,l})_\epsilon)_k\Big) + \sum_{1\le j\le n}(M_{\epsilon,\epsilon})_{i,j}((M^{\omega,l})_\epsilon)_j \\
&= \sum_{1\le j,k\le n}(M_{\epsilon,y_k})_{i,j}((M^{\omega,l})_{y_k})_j + \sum_{1\le j\le n}(M_{\epsilon,\epsilon})_{i,j}((M^{\omega,l})_\epsilon)_j \\
&= \sum_{1\le k\le n}(M_{\epsilon,y_k}(M^{\omega,l})_{y_k})_i + (M_{\epsilon,\epsilon}(M^{\omega,l})_\epsilon)_i \\
&= ((MM^{\omega,l})_\epsilon)_i = ((M^{\omega,l})_\epsilon)_i, \qquad \text{for each } 1\le i\le n\,.
\end{aligned}$$

The first equality is by Lemma 13, the last equality by Theorem 11(i). The result follows. ◀

The following is now immediate by Theorem 15 and our previous discussion.

▶ **Corollary 16.** *Let $r \in S^{alg}\langle\langle\Sigma^*\rangle\rangle \times S^{alg}\langle\langle\Sigma^\omega\rangle\rangle$ such that r is a component of a solution of an ω-algebraic system in Greibach normal form. Then there exists a simple ω-reset pushdown automaton with behavior r.*

7 Discussion

We have extended the characterization of ω-algebraic series so that we can use the ω-Kleene closure to transfer the property of Greibach normal form from algebraic systems to mixed ω-algebraic ones. This generalizes a fundamental property from context-free languages.

We believe that the same technique can be used to transfer other properties of algebraic systems to infinite words. Cohen, Gold [4] use this technique also for the elimination of chain rules, for the Chomsky normal form and for effective decision methods of emptiness, finiteness and infiniteness.

The second part transforms ω-algebraic series into simple ω-reset pushdown automata. Simple ω-reset pushdown automata do not use ϵ-transitions; in the literature, this is also called a *realtime* pushdown automaton. Realtime pushdown automata read a symbol of the input word in every transition - exactly like context-free grammars in Greibach normal form generate a letter in every derivation step. Additionally, each derivation step of context-free grammars in Greibach normal form increases the number of non-terminals in the sentential form by at most one. We showed that for realtime pushdown automata it suffices to handle at most one stack symbol per transition. Here the Greibach normal form provides exactly the properties needed to construct simple ω-reset pushdown automata.

As the first part applies only to mixed ω-algebraic systems, we could not use this result in the second part where the Greibach normal form is needed for ω-algebraic systems.

The model of simple ω-reset pushdown automata seems to be very natural. They occur when applying general homomorphisms to nested-word automata [1]. Their unweighted counterparts have been used for a Büchi-type logical characterization of timed pushdown languages [11] and ω-context-free languages [6]. A corresponding result for weighted ω-context-free languages is currently in development and uses the simple ω-reset pushdown automata introduced here.

References

1 A. Blass and Y. Gurevich. A note on nested words. *Microsoft Research*, 2006. URL: https://www.microsoft.com/en-us/research/publication/180-a-note-on-nested-words/.

2 S. L. Bloom and Z. Ésik. *Iteration Theories.* EATCS Monographs on Theoretical Computer Science. Springer, 1993. doi:10.1007/978-3-642-78034-9.

3 N. Chomsky and M. P. Schützenberger. The Algebraic Theory of Context-Free Languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier, 1963. doi:10.1016/S0049-237X(08)72023-8.

4 R. S. Cohen and A. Y. Gold. Theory of ω-Languages I: Characterizations of ω-Context-Free Languages. *Journal of Computer and System Sciences*, 15(2):169–184, 1977. doi:10.1016/S0022-0000(77)80004-4.

5 J. H. Conway. *Regular Algebra and Finite Machines.* Chapman & Hall, 1971.

6 M. Droste, S. Dziadek, and W. Kuich. Logic for ω-Pushdown Automata. *Information and Computation*, 2019. Special issue on "Weighted Automata", Accepted for publication.

7 M. Droste, S. Dziadek, and W. Kuich. Weighted Simple Reset Pushdown Automata. *Theoretical Computer Science*, 777:252–259, 2019. doi:10.1016/j.tcs.2019.01.016.

8 M. Droste, Z. Ésik, and W. Kuich. The Triple-Pair Construction for Weighted ω-Pushdown Automata. *Electronic Proceedings in Theoretical Computer Science*, 252:101–113, 2017. doi:10.4204/EPTCS.252.12.

9 M. Droste and W. Kuich. A Kleene Theorem for Weighted ω-Pushdown Automata. *Acta Cybernetica*, 23:43–55, 2017. doi:10.14232/actacyb.23.1.2017.4.

10 M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata.* EATCS Monographs in Theoretical Computer Science. Springer, 2009. doi:10.1007/978-3-642-01492-5.

11 M. Droste and V. Perevoshchikov. A Logical Characterization of Timed Pushdown Languages. In *10th International Computer Science Symposium in Russia (CSR 2015)*, volume 9139 of *LNCS*, pages 189–203. Spinger, 2015. doi:10.1007/978-3-319-20297-6_13.

12 S. Eilenberg. *Automata, Languages, and Machines*, volume 59, Part A of *Pure and Applied Mathematics.* Elsevier, 1974. doi:10.1016/S0079-8169(08)60880-6.

13 C. C. Elgot. Matricial theories. *Journal of Algebra*, 42(2):391–421, 1976. doi:10.1016/0021-8693(76)90106-X.

14 Z. Ésik and W. Kuich. A Semiring-Semimodule Generalization of ω-Context-Free Languages. In J. Karhumäki, H. Maurer, G. Păun, and G. Rozenberg, editors, *Theory Is Forever*, volume 3113 of *LNCS*, pages 68–80. Springer, 2004. doi:10.1007/978-3-540-27812-2_7.

15 Z. Ésik and W. Kuich. A Semiring-Semimodule Generalization of ω-Regular Languages II. *Journal of Automata, Languages and Combinatorics*, 10(2–3):243–264, 2005. doi:10.25596/jalc-2005-243.

16 Z. Ésik and W. Kuich. Modern Automata Theory, 2007. URL: http://www.dmg.tuwien.ac.at/kuich.

17 Z. Esik and W. Kuich. On Iteration Semiring-Semimodule Pairs. *Semigroup Forum*, 75(1):129–159, 2007. doi:10.1007/s00233-007-0709-7.

18 S. A. Greibach. A New Normal-Form Theorem for Context-Free Phrase Structure Grammars. *J. ACM*, 12(1):42–52, 1965. doi:10.1145/321250.321254.

19 W. Kuich. Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*, chapter 9, pages 609–677. Springer, 1997. doi:10.1007/978-3-642-59136-5_9.

20 W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs on Theoretical Computer Science.* Springer, 1986. doi:10.1007/978-3-642-69959-7.

21 A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series.* Texts and Monographs in Computer Science. Springer, 1978. doi:10.1007/978-1-4612-6264-0.

Transformations of Boolean Functions

Jeffrey M. Dudek
Department of Computer Science, Rice University, Houston, TX, USA
jmd11@rice.edu

Dror Fried
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel
dfried@openu.ac.il

Abstract

Boolean functions are characterized by the unique structure of their solution space. Some properties of the solution space, such as the possible existence of a solution, are well sought after but difficult to obtain. To better reason about such properties, we define *transformations* as functions that change one Boolean function to another while maintaining some properties of the solution space. We explore transformations of Boolean functions, compactly described as Boolean formulas, where the property is to maintain is the number of solutions in the solution spaces. We first discuss general characteristics of such transformations. Next, we reason about the computational complexity of transforming one Boolean formula to another. Finally, we demonstrate the versatility of transformations by extensively discussing transformations of Boolean formulas to "blocks," which are solution spaces in which the set of solutions makes a prefix of the solution space under a lexicographic order of the variables.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Boolean Formulas, Boolean Functions, Transformations, Model Counting

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.39

Funding *Jeffrey M. Dudek*: Supported by NSF grants DMS-1547433 and IIS-1527668, by the Big-Data Private-Cloud Research Cyberinfrastructure MRI-award funded by NSF under grant CNS-1338099, by the Ken Kennedy Institute Computer Science & Engineering Enhancement Fellowship funded by the Rice Oil & Gas HPC Conference, and by the Ken Kennedy Institute 2017/18 Cray Graduate Fellowship.

Acknowledgements We would like to thank Kuldeep S. Meel, Moshe Y. Vardi, and Rice's Computer-Aided Verification Group for useful discussions.

1 Introduction

Boolean functions play an integral part in many areas in computer science, electrical engineering and more [22, 11]. For example by abstracting properties of a system as a true/false dichotomy, one can model such properties as a Boolean function where a positive (true) output of that formula means that the property appears in the system. Typically every Boolean function can be uniquely characterized by its *solution space*, also called a *truth table*, which is a table that assigns the true/false output of the function for every possible assignment to the Boolean inputs. Since the size of such a table can be very large, in particular exponential in the number of variables, more compact representations of Boolean functions are used, such as Boolean formulas, Karnaugh maps [15], and Boolean Decision Diagrams (BDDs) [8]. Such compact representations, however, come at a cost since reasoning about properties of the Boolean function such as whether a solution exists, or counting the number of solutions, becomes a challenging problem. A question to ask, therefore, is whether one can better reason about properties of a Boolean function by "transforming" the function to a different Boolean function while still preserving some of the original properties.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 39; pp. 39:1–39:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this work, we lay foundations for thematic exploration of such transformations of Boolean functions. In our setting we use Boolean formulas to describe Boolean functions, and the property of the solution space that we maintain is the number of solutions in the solution space. In general, counting the number of solutions is a problem of great importance [12, 5, 16], which is known to be a #P-hard problem [21] and there are numerous works, both theoretical [20, 13] and applied [19, 17] that address this problem. In all this, the structure of the solution space can play an important role in the attempts for obtaining efficient counting [9, 7, 2].

In our formulation, transformations are quantified Boolean formulas that describe bijections between the output columns of solution spaces with the same number of variables. Thus the result of applying such a transformation T to a Boolean formula φ is a Boolean formula ψ with the same number of variables and same number of solutions as φ has.

This paper can be separated into three parts. In the first part we define transformations and discuss properties of transformations such as closure under composition and the inverse operation. We ask whether every pair of Boolean formulas with the same number of variables always has an expressible, polynomially-sized transformation between them. We discuss this in the second part and give an affirmative answer if the number of alternating quantifiers is not limited. Moreover, we show that if the number of alternations is bounded then the question is equivalent to the collapse of the polynomial hierarchy.

In the third part of the paper we present various transformations and combination of transformations that demonstrate the versatility of our framework. For that, we focus on a specific solution space structure called a “block,” in which the set of solutions form a prefix of the solution space, under a lexicographic order of the variables. Boolean formulas with such a block-type solution space– for example, chain formulas [9]– have efficient counting. We describe several techniques to construct transformations that merge solutions together in order to transform a solution space to a block. Specifically, we describe a method to, for an arbitrary given Boolean formula φ, construct a transformation (possibly of exponential size) that can transform φ to a block. We then present classes of transformations that can transform certain specialized types of solution spaces into blocks. The transformations in this part are “parameterized,” in the sense that there are certain parameters for the transformations that are adjusted according to the given Boolean formula.

Finally, a divide-and-conquer approach for a solution is a general technique also used in studies of Boolean functions [14, 10]. We show a transformation that can efficiently transform specific Boolean formulas to a block by first finding transformations to blocks for each sub-formula in a divide-and-conquer manner.

2 Preliminaries

A *Boolean function* $f : \{0,1\}^n \to \{0,1\}$ is a function that assigns n input Boolean variables $\{x_1, \cdots, x_n\}$ to a Boolean output 0 (*false*) or 1 (*true*). The *solution space*, also called a *truth table*, of f is the explicit description of f as a table of a size 2^n. A *solution* however is an assignment $\vec{\sigma}$ for which $f(\sigma) = 1$. Thus the solution space of f contains solutions and non-solutions.

Throughout this work, we generally assume that n is fixed and use $\vec{x}$ to denote a sequence of n variables $x_n, \cdots, x_1$. Moreover, we fix the order of the sequence of variables, thus have a lexicographic order $\leq_{lex}$ on the truth assignments to $\vec{x}$, where x_n is the most significant bit (msb) and x_1 the least significant bit (lsb). We also define a function

$bin : \{0, \cdots, 2^n - 1\} \to \{0,1\}^n$ that maps the integers $0 \leq c \leq 2^n - 1$ to their natural encoding as a corresponding assignment to n variables (e.g. for $n = 4$, $bin(3) = 0011$), and the corresponding inverse function $|\cdot| : \{0,1\}^n \to \{0, \cdots, 2^n - 1\}$ (then $|0011| = 3$).

One compact description of a Boolean function f is by a *Boolean formula* $\varphi_f(x_1, \cdots x_n)$ that is satisfied exactly when f is 1. When f is obvious or irrelevant, we omit f from the notation of φ_f. Every Boolean formula φ over n variables describes a unique solution space of size 2^n. The number of solutions to φ, i.e. the number of assignments that set φ to *true* is denoted by $\#\varphi$. The *size* of φ is defined as the length of φ and is denoted by $|\varphi|$. Two Boolean formulas φ, ψ that describe the same Boolean function are called *logically equivalent*, denoted $\varphi \equiv \psi$, and by definition such formulas have identical solution spaces.

In our settings, we also reason about partial solution spaces (also called *subspaces*). For a solution space S, the partial solution $S[\sigma_n, \cdots, \sigma_{i+1}]$ of size 2^i is obtained by assigning $\sigma_j \in \{0,1\}$ to the variable x_j for each j such that $i + 1 \leq j \leq n$. We denote by

$$S[x_n, \cdots, x_{i+1}] = \{S[\sigma_n, \cdots, \sigma_{i+1}] \mid (\sigma_n, \cdots \sigma_{i+1}) \in \{0,1\}^{n-i}\}$$

the set of all solution spaces of size 2^i obtained by fixing $x_n, \cdots x_{i+1}$ to every assignment. When $\vec{\sigma}$ is a complete assignment for the variables $x_n \cdots x_1$, then $S[\vec{\sigma}]$ denotes the value of the assignment $\vec{\sigma}$ in S (i.e. *true* or *false*). A *null* solution space is a solution space in which all its assignments are valued to 0.

A *block* is a solution space whose set of solutions make a prefix under the fixed lexicographic order $\leq_{lex}$. That is, a (possibly partial) solution space S of size 2^i is a block if, for every pair of assignments $\vec{\sigma}, \vec{\sigma}' \in \{0,1\}^i$ where $S[\vec{\sigma}] = 1$ and $\vec{\sigma}' \leq_{lex} \vec{\sigma}$, it holds that $S[\vec{\sigma}'] = 1$. We can also describe a block S by its output column as $1^k 0^{2^i - k}$ for some positive integer k. In this case, k is exactly the number of solutions in S. Finally for every $0 \leq c \leq 2^n$, we define $block_c$ to be the formula over n variables whose solution space is the block with c solutions. That is, $block_c(\vec{x}) \equiv (\vec{x} \leq_{lex} bin(c)) \land (\vec{x} \neq bin(c))$. Note that $block_c$ can be written as a Boolean formula (in particular, as a *chain formula* [9]). When obvious from the context we sometimes refer to the formula $block_c$ as a block with c solutions.

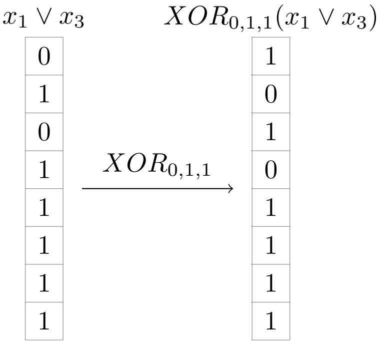
3 Definitions and properties

Given a function $g : \{0,1\}^n \to \{0,1\}^m$ for some integers n, m, we say that a quantified Boolean formula $F(\vec{x}, \vec{y})$ *describes* g if $\vec{x}$ (called the *domain variables*) is of size n, $\vec{y}$ (called the *range variables*) is of size m, and for every assignments $\vec{a}, \vec{b}$ for $\vec{x}, \vec{y}$ respectively we have $F(\vec{a}, \vec{b}) = true$ iff $g(\vec{a}) = \vec{b}$.

We now define transformations as follows:

▶ **Definition 1.** *A* transformation $T(\vec{x}, \vec{y})$ *is a quantified Boolean formula over* $2n$ *free variables that describes a bijection from* $\{0,1\}^n$ *to* $\{0,1\}^n$.

We assume without loss of generality that all transformations are described in a prenex normal form. The *size* of a transformation T is the number of symbols in the underlying QBF formula, denoted $|T|$. Although we allow arbitrary alternation of quantifiers in transformations, one might consider restricting to transformations in Σ_k^P (for some fixed k) in order to limit the number of quantifier alternations and hence ease reasoning. In particular, restricting to transformations in Σ_1^P (i.e. using only existential quantifiers), allows reasoning on such transformations by using SAT solvers, while still maintaining some expressiveness, e.g. by using "carry" bits as we see in Section 5.1. Unless mentioned otherwise, for the rest of the paper we assume that all the Boolean formulas have n free variables and all transformations have $2n$ free variables.



$x_1 \vee x_3$	$XOR_{0,1,1}(x_1 \vee x_3)$
0	1
1	0
0	1
1	0
1	1
1	1
1	1
1	1

Figure 1 The truth tables of the formula $x_1 \vee x_3$ and of the result after applying the transformation $XOR_{0,1,1}$. Each truth table is given in lexicographic order from top (where $x_3 = x_2 = x_1 = 0$) to bottom (where $x_3 = x_2 = x_1 = 1$).

We now define how to apply a transformation to a Boolean formula. Given a transformation $T(\vec{x}, \vec{y})$ that describes a bijection $g : \{0,1\}^n \to \{0,1\}^n$ and a Boolean formula $\varphi(\vec{z})$ over n free variables, we apply T to φ by constructing a new Boolean formula $Apply_{T,\varphi}$ over n free variables in which we identify the domain variables of T with the variables of φ:

$$Apply_{T,\varphi}(\vec{y}) \equiv \exists \vec{x}\ (T(\vec{x}, \vec{y}) \wedge \varphi(\vec{x})).$$

For convenience, we denote $Apply_{T,\varphi}$ by $T(\varphi)$. If ψ is a Boolean formula over n variables and $\psi \equiv T(\varphi)$, we say that T *transforms* φ *into* ψ. Note that we can also think of a transformation as a function from Boolean formulas to Boolean formulas. Also notice that $T(\varphi)$ is a Boolean formula with n free variables whose solution space is resulted by applying g (i.e., the bijection described by T) to the solution space of φ. That is, $\vec{a}$ is a solution of φ if and only if $g(\vec{a})$ is a solution of $T(\varphi)$. Since g is a bijection, it follows that φ and $T(\varphi)$ indeed have the same number of solutions. Although transformations are defined over formulas, to ease the reading we sometimes say that we apply transformations to a solution space, in which case we mean that we apply the transformation to a formula with the mentioned solution space.

▶ **Example 2.** The simplest transformation is the *identity* transformation $id(\vec{x}, \vec{y}) \equiv \bigwedge_i x_i \leftrightarrow y_i$. For all Boolean formulas φ, $id(\varphi) \equiv \varphi$.

▶ **Example 3.** The transformation $T_1(x_1, x_2, y_1, y_2) \equiv (x_1 \leftrightarrow y_2) \wedge (x_2 \leftrightarrow y_1)$ is a transformation over 4 free variables. When applied to a Boolean formula $\varphi(x_1, x_2)$, T_1 syntactically switches between x_1 and x_2. That is, $T_1(\varphi(x_1, x_2)) \equiv \varphi(x_2, x_1)$.

▶ **Example 4.** For a given vector $\vec{a} \in \{0,1\}^n$, the bijection that maps every $\vec{b} \in \{0,1\}^n$ to $\vec{b} \oplus \vec{a}$ is represented by the XOR transformation:

$$XOR_{\vec{a}}(\vec{x}, \vec{y}) \equiv \bigwedge_i (y_i \leftrightarrow (x_i \oplus a_i)).$$

Figure 1 describes an example of the XOR transformation.

3.1 Properties of transformations

We consider transformations as combinatorial objects that can be used to construct other, more complicated transformations. For that, we describe a few simple algebraic properties of transformations and then define the composition of two transformations.

We begin by listing a few simple properties of transformations, which follow directly from the fact that transformations describe bijections:

▷ Claim 5. Let T be a transformation and let φ and ψ be Boolean formulas. Then:

1. $T(\varphi \vee \psi) \equiv T(\varphi) \vee T(\psi)$
2. $T(\varphi \wedge \psi) \equiv T(\varphi) \wedge T(\psi)$
3. $T(\neg\varphi) \equiv \neg(T(\varphi))$

We next consider compositions of transformations. Let T_1 and T_2 be transformations that describe bijections g_1 and g_2. We define the *composition* of T_1 and T_2, denoted $T_2 \circ T_1$, to be a transformation that describes the composition of g_1 and g_2 (which is the bijection $g_2 \circ g_1$ that maps each $\vec{a} \in \{0,1\}^n$ to $g_2(g_1(\vec{a}))$). Note that composition can be described by a simple syntactic construction of $T_2 \circ T_1$ from T_1 and T_2 as the following claim shows.

▷ Claim 6. $(T_2 \circ T_1)(\vec{x}, \vec{y}) \equiv \exists \vec{z}\ (T_1(\vec{x}, \vec{z}) \wedge T_2(\vec{z}, \vec{y}))$

Naturally, we also have that for an arbitrary Boolean formula φ, applying $T_2 \circ T_1$ to φ is logically equivalent to applying T_1 to φ followed by applying T_2:

▷ Claim 7. $(T_2 \circ T_1(\varphi))$ is logically equivalent to $T_2(T_1(\varphi))$.

Proof. We have that:

$$\begin{aligned} T_2(T_1(\varphi))(\vec{z}) &\equiv \exists \vec{y}\ (T_2(\vec{y}, \vec{z}) \wedge T_1(\varphi)(\vec{y})) \\ &\equiv \exists \vec{y}\ (T_2(\vec{y}, \vec{z}) \wedge \exists \vec{x}\ (T_1(\vec{x}, \vec{y}) \wedge \varphi(\vec{x}))) \\ &\equiv \exists \vec{x}\ (\exists \vec{y}\ (T_1(\vec{x}, \vec{y}) \wedge T_2(\vec{y}, \vec{z})) \wedge \varphi(\vec{x})) \equiv (T_2 \circ T_1)(\varphi)(\vec{z}) \end{aligned}$$ ◁

Notice that if T_1 and T_2 are both in Σ_k^P for some k, then Claim 6 proves that their composition $T_2 \circ T_1$ is in Σ_k^P as well. We will specifically use this fact for the merge-rotation transformations described in Section 5.2 which are in Σ_1^P. A transformation constructed by composition of many Σ_1^P transformations is also in Σ_1^P and so can still be reasoned about with a SAT solver.

Along the same lines, we define the *inverse transformation* of a transformation T (that describes a bijection g) to be a transformation T^{-1} that describes the inverse bijection g^{-1}. As with composition, there is a simple syntactic construction of T^{-1} from T by swapping the domain and range variables:

▷ Claim 8. $T^{-1}(\vec{x}, \vec{y}) \equiv T(\vec{y}, \vec{x})$

Proof. Let g be the bijection described by T. Notice that for every assignment $\vec{a}$ and $\vec{b}$ to $\vec{x}$ and $\vec{y}$ we have that $T(\vec{b}, \vec{a}) = \mathit{true}$ if and only if $g(\vec{b}) = \vec{a}$, which occurs if and only if $g^{-1}(\vec{a}) = \vec{b}$. Hence $T(\vec{y}, \vec{x})$ indeed describes g^{-1}. ◁

As a direct result from Claim 8, we get that the inverse transformation is indeed an inverse under the composition operator as follows.

▶ **Corollary 9.** *Let T be a transformation and φ be a Boolean formula. Then $(T \circ T^{-1})(\varphi) \equiv (T^{-1} \circ T)(\varphi) \equiv \varphi$.*

4 Transformations and the polynomial hierarchy

In this work the transformations that we define do not affect the number of solutions of a formula. In particular, if a transformation transforms a Boolean formula φ_1 into φ_2 then the number of solutions of φ_1 and φ_2 must be the same. Perhaps surprisingly, the converse is also true: if two Boolean formulas φ_1 and φ_2 over the same number of variables n have the same number of solutions then there must be a transformation of size polynomial in $max\{n, |\varphi_1|, |\varphi_2|\}$ that transforms φ_1 into φ_2. We give this result as the following theorem.

▶ **Theorem 10.** *There is a polynomial $p : \mathbb{N}^3 \to \mathbb{N}$ such that, if φ_1 and φ_2 are two Boolean formulas with n variables each and the same number of solutions, then there is a transformation T of size no more than $p(n, |\varphi_1|, |\varphi_2|)$ such that $T(\varphi_1) \equiv \varphi_2$.*

Proof. For a given Boolean formula φ of n variables and an assignment $\{0,1\}^n$, let $H^1_\varphi(\vec{a})$ be the set of solutions strictly smaller, under $\leq_{lex}$ than $\vec{a}$. That is $H^1_\varphi(\vec{a}) = \{\vec{c} \mid \vec{c} <_{lex} \vec{a} \wedge \varphi(\vec{c}) = 1\}$. Similarly, let $H^0_\varphi(\vec{a}) = \{\vec{c} \mid \vec{c} <_{lex} \vec{a} \wedge \varphi(\vec{c}) = 0\}$ be the set of non-solutions strictly smaller than $\vec{a}$.

Now let $\mathcal{L} \subseteq \{0,1\}^n \times \{0,1\}^n$ be the set of $(\vec{a}, \vec{b}) \in \{0,1\}^n \times \{0,1\}^n$ such that either: (i) $\varphi_1(\vec{a}) = \varphi_2(\vec{b}) = 1$, and $|H^1_{\varphi_1}(\vec{a})| = |H^1_{\varphi_2}(\vec{b})|$, or; (ii) $\varphi_1(\vec{a}) = \varphi_2(\vec{b}) = 0$, and $|H^0_{\varphi_1}(\vec{a})| = |H^0_{\varphi_2}(\vec{b})|$.

Since for an arbitrary $\vec{a} \in \{0,1\}^n$ and an arbitrary Boolean formula φ, both $|H^1_\varphi(\vec{a})|$ and $|H^0_\varphi(\vec{a})|$ can be computed by using a single $\#P$ query, $\mathcal{L}$ belongs to $P^{\#P}$ and consequently to $PSPACE$. Therefore there is a polynomially-sized QBF formula T with $2n$ free variables whose solution space is $\mathcal{L}$.

Finally, recall that φ_1 and φ_2 have the same number of solutions. For every $\vec{a} \in \{0,1\}^n$, there is therefore exactly one $\vec{b} \in \{0,1\}^n$ such that $(\vec{a}, \vec{b}) \in \mathcal{L}$. Thus T describes a bijection and so T is indeed a transformation. Together with the fact that, by definition $\varphi_1(\vec{a}) = \varphi_2(\vec{b})$ for every $(\vec{a}, \vec{b}) \in \mathcal{L}$, we have that that $T(\varphi_1) \equiv \varphi_2$. ◀

The transformation T obtained by Theorem 10 may have arbitrarily nested quantifiers. A natural question to ask is whether it is possible to generalize Theorem 10 while limiting the number of alternating quantifiers. This leads us to the following conjecture, which restricts the number of quantifier alternations to some $k \geq 1$.

▶ **Conjecture 1** (Transformation Conjecture at k). *There is an integer $k \geq 1$ and a polynomial $p : \mathbb{N}^3 \to \mathbb{N}$ such that, if φ_1 and φ_2 are two Boolean formulas with n variables each and the same number of solutions, then there is a transformation $T \in \Sigma^P_k$ of size no more than $p(n, |\varphi_1|, |\varphi_2|)$ such that $T(\varphi_1) \equiv \varphi_2$.*

As we next show via the following two lemmas, our conjecture is equivalent to open unsolved questions in computational complexity.

We first generalize our proof of Theorem 10 to the restricted setting of the conjecture. In order to obtain Σ^P_k transformations, our proof requires the stronger, open assumption that the polynomial hierarchy collapses at or before Σ^P_k (in place of the fact used in Theorem 10 that $P^{\#P} \subseteq PSPACE$). We state this result as the following lemma.

▶ **Lemma 11.** *For all $k \geq 1$, if $P^{\#P} \subseteq \Sigma^P_k$ then the Transformation Conjecture at k holds.*

Proof. Let Boolean formulas φ_1 and φ_2 be Boolean formulas over n variables each, with the same number of solutions. Consider the language $\mathcal{L} \subseteq \{0,1\}^n \times \{0,1\}^n$ defined in the proof of Theorem 10. In particular, if $P^{\#P} \subseteq \Sigma^P_k$ then $\mathcal{L}$ belongs to Σ^P_k. It follows that there is a polynomially-sized Σ^P_k formula T with $2n$ free variables whose solution space is L. Hence, as in the proof of Theorem 10, T is a transformation and $T(\varphi_1) \equiv \varphi_2$. ◀

We next show in Lemma 12 that on the other hand the Transformation Conjecture implies that the polynomial hierarchy collapses at or before the level Σ_{k+4}^P. Tightening the result to prove the exact converse of Lemma 11, which would be that the Transformation Conjecture implies the collapse of the polynomial hierarchy at or before Σ_k^P, remains for future work.

▶ **Lemma 12.** *For all $k \geq 1$, if the Transformation Conjecture at k holds then $P^{\#P} \subseteq \Sigma_{k+4}^P$.*

Proof. Let $p : \mathbb{N}^3 \to \mathbb{N}$ be the polynomial from the conjecture. It suffices to prove that, given a Boolean formula φ over n variables and an index $1 \leq i \leq n+1$, we can construct (in polynomial time) a Σ_{k+4}^P Turing Machine M that takes φ and i as input and accepts if and only if the i-th bit of $\#\varphi$ is 1. This decision problem is complete for $P^{\#P}$.

Our Turing Machine M first guesses a formula $T \in \Sigma_k^P$ (in prenex normal form) over $2n$ variables and an integer $0 \leq c \leq 2^n$ and makes a sequence of Π_{k+3}^P queries to verify that: (1) T is a transformation, (2) T transforms φ into a block with c solutions, and (3) the i-th bit of c is 1. M accepts if and only if these three conditions hold for some guess T and c, where T has size no more than $p(n, |\varphi|, |block_c|)$ and $0 \leq c \leq 2^n$.

Intuitively, c is our verified guess of the number of solutions for φ, so that M can check if the i-th bit of $\#\varphi$ is 1 just by consulting c. The transformation T is used to verify c.

The first property (that T is a transformation) can be verified by making a Π_{k+3}^P query followed by a Π_{k+1}^P query:

$$\alpha(T) \equiv \forall \vec{x}\ \exists \vec{y}\ \forall \vec{z}\ (T(\vec{x}, \vec{z}) \leftrightarrow (\vec{y} = \vec{z}))$$
$$\beta(T) \equiv \forall \vec{y}\ \exists \vec{x}\ (T(\vec{x}, \vec{y})).$$

In particular, $\alpha(T)$ evaluates to true if and only if T describes some function g, and $\beta(T)$ then evaluates to true if and only if g is invertible. Thus $\alpha(T)$ and $\beta(T)$ together hold if and only if T describes a bijection g, i.e. if and only if T is a transformation.

The second property (that T transforms φ into a block with c solutions) can be verified by a single Π_{k+1}^P query:

$$\gamma(T, c) \equiv \forall \vec{x}\ \exists \vec{y}\ (T(\vec{x}, \vec{y}) \wedge (\varphi(\vec{x}) \leftrightarrow block_c(\vec{y})))$$

Recall from Section 2 that, for $0 \leq c \leq 2^n$, $block_c(\vec{x}) \equiv (\vec{x} \leq bin(c)) \wedge (\vec{x} \neq bin(c))$ is the Boolean formula whose solution space is a block with c solutions.

Finally, the third property (that the i-th bit of c is 1) can be verified simply by reading the bits of c.

Since M makes a single polynomially-sized guess (of T and c) followed by three Π_k^P queries, M is indeed a Σ_{k+4}^P Turing Machine. Since transformations preserve the number of solutions and $\#block_c = c$, then if M accepts it means that φ must have c solutions. Moreover, if M accepts then the i-th bit of c is 1. Thus the i-th bit of $\#\varphi$ is indeed 1. Conversely, consider the case where the i-th bit of $\#\varphi$ is 1. By the Transformation Conjecture there exists a polynomially-sized transformation $T' \in \Sigma_k^P$ that transforms φ into $block_{\#\varphi}$. Thus M will accept with $T = T'$ and $c = \#\varphi$. ◀

5 Transformations to blocks

In this section we give examples of how to use the transformations definitions and properties defined in Section 3 to construct and combine various transformations in order to manipulate the solution space to a specific structure. For that, we choose the structure of a block solution space and we focus on a specific type of transformations that transform a given formula into a block.

In general, Boolean formulas with a block solution space, such as $block_c$ or chain formulas [9] have efficient counting by a simple binary-search method. To see this, assume that φ is a formula with n variables and a block solution space. Then for every assignment $\vec{\sigma}$ to the variables of φ, we have that $\varphi(\vec{\sigma}) = 1$ if and only if $\#\varphi \geq |\vec{\sigma}|$. Therefore $\#\varphi$ can be found by at most n such queries. Thus we have that a hypothetical efficient transformation of a given formula to a block can lead to efficient counting.

Moreover, in the setting of transformations, the following claim, which follows directly from the transformation properties discussed in Section 3, shows that by exploring transformations to blocks we can also get a better understanding on transformations between every two formulas.

▷ Claim 13. Let φ_1, φ_2 be Boolean formulas with the same number of solutions, and with transformations T_1, T_2 respectively to blocks . Then $T_2^{-1} \circ T_1(\varphi_1) \equiv \varphi_2$.

Proof. Assume that $\#\varphi_1 = \#\varphi_2 = c$. Then the transformations T_1, T_2 transform φ_1 and φ_2 respectively to a block of c solutions. Then $T_1(\varphi_1) \equiv T_2(\varphi_2) \equiv block_c(\vec{x})$. Then from the transformation properties we have that $T_2^{-1}(block_c(\vec{x})) \equiv \varphi_2$, thus $T_2^{-1} \circ T_1(\varphi_1) \equiv \varphi_2$. ◁

We first describe a type of transformations, possibly of exponential size to the size of the input, that can block any Boolean formula. These transformations are based on merging the solutions in the solution space together. We then explore a different technique, more efficient size-wise, of transformations, called *merge-rotate* that merges subspaces, that are already blocks, into a single block. We show how by iterating the merge-rotate transformations we can transform more sophisticated solution spaces to a block. Finally we demonstrate the use of the iterative approach to efficiently transform a specific type of formulas that are conjunction of two variable-disjoint sub-formulas, once their transformations to blocks are found. The transformations that we describe here are "parameterized" in the sense that the transformations use additional parameters that are depended on the given input formula. Exploring so called "oblivious" transformations that do not have such parameters is left for future work.

5.1 Transforming general formulas to blocks

A general description of a solution space S for every Boolean formula φ is as a sequence of alternating intervals of all solutions and all non-solutions. That is, $S = (1^{k_1}0^{k_2} \cdots 1^{k_{\ell-1}}0^{k_\ell})$ where $0 \leq k_i \leq 2^n$ for every $i \leq \ell$ for some even ℓ, and $\sum_i k_i = 2^n$. In this section, we show a general Σ_1^P transformation, of size polynomial in $\max\{\ell, n\}$, that blocks S.

For that, we first describe *addition* as a way to "shift" whole intervals in a solution space. Let $\psi_{+,i}(\vec{y}, \vec{a}, \vec{b})$ be the following Σ_1^P formula:

$$\exists z_1 \cdots \exists z_i \left(\neg z_1 \wedge \bigwedge_{j=1}^{i} (y_j \leftrightarrow a_j \oplus b_j \oplus z_j) \wedge \bigwedge_{j=1}^{i-1} (z_{j+1} \leftrightarrow ((z_j \wedge a_j) \vee (z_j \wedge b_j) \vee (a_j \wedge b_j))) \right)$$

The $\vec{z}$ variables represent the carry in the addition of the $\vec{a}$ and $\vec{b}$ variables. Recall from Section 2 that $|\cdot| : \{0,1\}^n \to \{0, 1, \cdots, 2^n - 1\}$ produces the n-bit positive integer corresponding to an assignment. Then we have the following.

▷ Claim 14. For all integers $0 < i \leq n$ and assignments $\vec{y}, \vec{a}, \vec{b} \in \{0,1\}^n$, $\psi_{+,i}(\vec{y}, \vec{a}, \vec{b}) = true$ if and only if $|\vec{y}| = |\vec{a}| + |\vec{b}| \pmod{2^i}$.

Now let $interval_{(c,d)}(\varphi(x)) \equiv (bin(c) \leq_{lex} \vec{x} <_{lex} bin(d))$ be the formula that is true for every assignment $\vec{\sigma}$ over the n variables for which $|\vec{\sigma}| \in [c,d)$. Denote by $k'_i = \sum_{h \leq i} k_h$ the last index of the i'th interval and set $k'_0 = 0$. Then let $Merge_{(k_1,\cdots k_\ell)}(\vec{x},\vec{y})$ be the following transformation:

$$Merge_{(k_1,\cdots k_\ell)}(\vec{x},\vec{y}) = \bigwedge_{j=0}^{\ell/2-1} \Big(interval_{(k'_{2j},k'_{2j+1})}(\vec{x}) \to \psi_{+,n}(\vec{y},\vec{x},2^n - \sum_{1 \leq h \leq j} k_{2h}) \wedge \\ interval_{(k'_{2j+1},k'_{2j+2})}(\vec{x}) \to \psi_{+,n}(\vec{y},\vec{x},\sum_{j<h<\ell/2} k_{2h+1})\Big)$$

Note that $Merge_{(k_1,\cdots k_\ell)}(\vec{x},\vec{y})$ is of size polynomial in $\max\{\ell,n\}$.

▷ Claim 15. Let φ be a Boolean formula with a solution space described as $S = (1^{k_1}0^{k_2}\cdots 1^{k_{\ell-1}}0^{k_\ell})$ where $0 \leq k_i \leq 2^n$ for all $0 \leq i \leq \ell$ and $\sum_i k_i = 2^n$. Let $k = \sum_{i=0}^{\ell/2-1} k_{2i+1}$. Then $Merge_{(k_1,\cdots k_\ell)}$ is a Σ_1^P transformation that transforms φ to the block $(1^k 0^{2^n-k})$.

Proof. The transformation $Merge_{(k_1,\cdots k_\ell)}$ simply shifts the j-th odd interval (which are all 1) to be the j-th interval in the lexicographic order by shifting the interval past all earlier 0 blocks, and the j-th even interval (which are all 0) to be the $\ell/2 + j$-th interval in the lexicographic order by shifting the interval past all later 1 blocks. Thus all 0 blocks occur lexicographically after all 1 blocks following the transformation. ◁

When ℓ is exponential in n, the size of $Merge_{(k_1,\cdots k_\ell)}$ is exponential in n as well. In Sections 5.2 and 5.3 we explore ways to maintain efficient size transformations for certain solution spaces with an exponential number of intervals.

5.2 The merge-rotate transformation

We next turn our attention to a different technique of transformations to blocks called the *merge-rotate* transformation. For merge-rotate we assume that a given solution subspace B is "halved" into an "upper" subspace B_0 and a "lower" subspace B_1 that are already in block forms. The transformation merge-rotate (as its name implies) rotates B_0 in order to merge its solutions with B_1, then rotates the entire subspace B to turn B to a block. We describe the merge-rotate technique, then see how to extend merge-rotate to an iterative process that can handle more complicated solution spaces.

We start from the ψ_+ formula, defined in Section 5.1, upon which we define the following $\mathbf{rot}_c$ transformation for a given integer $0 \leq c < 2^n$.

▶ **Definition 16.** *For given $0 \leq c < 2^n$ and $0 < i \leq n$, let $\mathbf{rot}_{c,i}(\vec{x},\vec{y}) = \psi_{+,i}(\vec{y},\vec{x},bin(\vec{c})) \wedge \bigwedge_{j=i+1}^{n}(x_j \leftrightarrow y_j)$.*

By applying $\mathbf{rot}_{c,i}$ to a Boolean formula φ we "rotate" each subspace in $S[x_n,\cdots,x_{i+1}]$ of size 2^i of φ by c steps (mod 2^i).

Next let $\sigma_n,\cdots\sigma_{i+1}$ be an assignment to $x_n,\cdots,x_{i+1}$ and assume that the subspaces $B_0 = S[\sigma_n,\cdots\sigma_{i+1},0]$ and $B_1 = S[\sigma_n,\cdots\sigma_{i+1},1]$ are already in block forms, with number of solutions k and k' respectively. The overall subspace $B = S[\sigma_n,\cdots\sigma_{i+1}]$ has the form $(1^k 0^{2^{i-1}-k} 1^{k'} 0^{2^{i-1}-k'})$. We show how to use the rotation transformation on these blocks, to transform the subspace B a block of the form $(1^{k+k'} 0^{2^i-(k+k')})$. For that, we need to restrict the rotation only to B_0 in order to merge the solutions of B_0 and B_1. We then use rotation on the entire subspace B to rotate B to a block form.

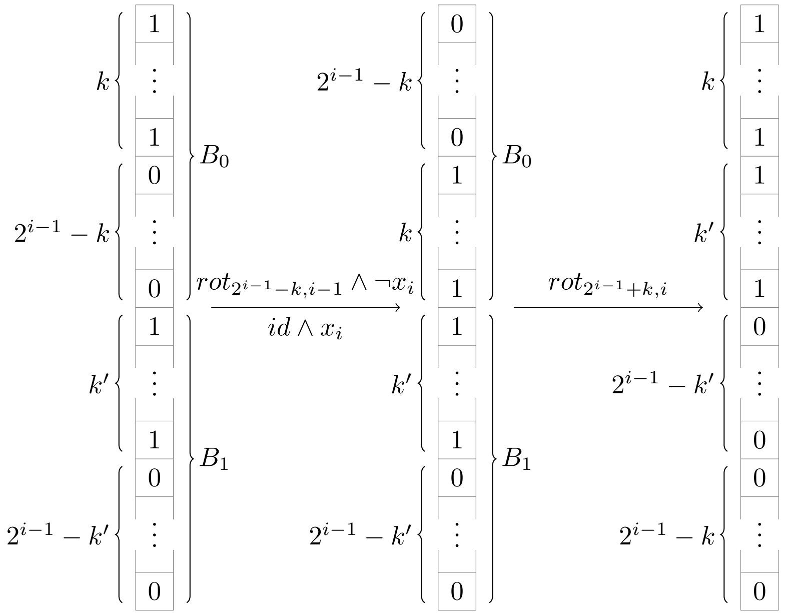


Figure 2 The transformation to blocks $MergeRotate$ is a composition of two rot transformations. Each solution subspace of the form $B = (1^k 0^{2^{i-1}-k} 1^{k'} 0^{2^{i-1}-k'})$ is transformed on B_0 by $rot_{2^{i-1}-k,i-1}$ and on B_1 by the identity transformation to $(0^{2^{i-1}-k} 1^{k+k'} 0^{2^{i-1}-k'})$ and then by $rot_{2^{i-1}-k,i}$ to $(1^{k+k'} 0^{2^{i-1}-(k+k')})$.

This results in the following transformation which we call *MergeRotate*, also depicted in Figure 2. Note that *id* is the identity transformation as defined in Section 3.

▶ **Definition 17.** *Let $k, i \leq n$ be given. The transformation MergeRotate is defined as:*

$$MergeRotate(k, i) \equiv rot_{2^{i-1}+k,i} \circ ((rot_{2^{i-1}-k,i-1} \wedge \neg x_i) \vee (id \wedge x_i))$$

Then we have the following claim, whose proof follows from the definition of $MergeRotate$.

▷ Claim 18. Let $B_0 = S[\sigma_n, \cdots \sigma_{i+1}, 0]$ and $B_1 = S[\sigma_n, \cdots \sigma_{i+1}, 1]$ for some index $i \leq n$ and $(\sigma_n, \cdots \sigma_{i+1}) \in \{0, 1\}^{n-i}$. Assume that B_0 and B_1 are blocks of size k and k' respectively. Then the transformation $MergeRotate(k, i)$ transforms $B = S[\sigma_n, \cdots \sigma_{i+1}]$ to a block of $k + k'$ solutions.

Note that k', the number of solutions in B_1, is not required for $MergeRotate$. Also note that $MergeRotate$ is in NP and in size polynomial to n.

We next give a technical lemma, which we make use of in Section 5.3, that shows that when using $MergeRotate$ we do not always need to have the rotation as the exact size of the block of B_0, as long as B_1 is a null solution space.

▶ **Lemma 19.** *Let $B_0 = S[\sigma_n, \cdots \sigma_{i+1}, 0]$ and $B_1 = S[\sigma_n, \cdots \sigma_{i+1}, 1]$ for some index $i \leq n$ and $(\sigma_n, \cdots \sigma_{i+1}) \in \{0, 1\}^{n-i}$. Assume that B_0 is a block of size k' and that B_1 is a null solution space (note that it means that the overall subspace $B = (1^{k'} 0^{2^{i-1}-k'} 0^{2^{i-1}})$ is already a block of k' solutions). Then for every $k' \leq k \leq 2^{i-1}$ the transformation $MergeRotate(k, i)$ maintains $B = S[\sigma_n, \cdots \sigma_{i+1}]$ as a block of k' solutions.*

Proof. Note that when applying $MergeRotate$, we first rotate only B_0 by $2^{i-1} - k$. This results in a space $B' = (0^{2^{i-1}-k} 1^{k'} 0^{k-k'} 0^{2^{i-1}})$. Now the second rotation rotates B' by $2^{i-1} + k$ which makes $B'' = (1^{k'} 0^{k-k'} 0^{2^{i-1}} 0^{2^{i-1}-k}) = (1^{k'} 0^{2^i - k'})$ as required. ◀

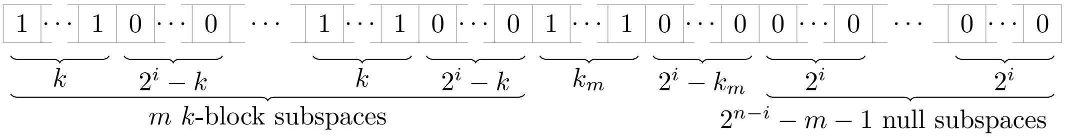


Figure 3 The truth table of a k-block i-suffix-null solution space, given in lexicographic order from left to right as described in Definition 20.

In the next sections we show where *MergeRotate* can be used iteratively to construct transformations of polynomial size to blocks for formulas with a specific solution space structure.

5.3 Iterating the merge-rotate transformations

Having defined the merge-rotate transformation, we would like to see how to use it to transform more complex solution spaces, with possibly an exponential number of intervals, into blocks. For that, a natural solution space that can demonstrate the iterative use of merge-rotate is a solution space S where, for some integer $0 \leq k \leq 2^i$ and every $(\sigma_n, \cdots \sigma_{i+1}) \in \{0,1\}^{n-1}$, the subspace $S[x_n, \cdots x_{i+1}]$ is a block of size k. In fact, we can make a somewhat stronger statement on a more complicated solution space structure as defined below. This structure also appears later in Section 5.4.

▶ **Definition 20.** *For a given $i \leq n$, and $0 \leq k \leq 2^i$, a solution space is said to be a k-block i-suffix-null if there exists an integer $0 \leq m \leq 2^{n-i}$ such that: (i) every solution space $S[\sigma_n, \cdots, \sigma_{i+1}]$ where $(\sigma_n, \cdots \sigma_{i+1}) >_{lex} bin(m)$ is a null solution space; (ii) every solution space $S[\sigma_n, \cdots, \sigma_{i+1}]$ where $(\sigma_n, \cdots \sigma_{i+1}) <_{lex} bin(m)$ is a block of size k; (iii) $S[bin(m)]$ is a block of size $0 \leq k_m \leq k$.*

Figure 3 depicts a k-block i-suffix-null solution space. Note that m can be 0 which means that the entire solution space is null, or can be 2^{n-i} in which all that the elements in $S[x_n, \cdots x_{i+1}]$ are k-blocks. Moreover, if $i = n$ then S is a block of size k.

We next show how to construct a transformation composed of $n - i$ merge-rotate transformations in order to block a k-block i-suffix-null solution space. For given $i < n$ and $k < 2^i$, let $ItrMergeRotate(k, i)$ be the following transformation:

$$MergeRotate(2^{n-i-1}k, n) \circ \cdots \circ MergeRotate(2^{j-1}k, i + j) \circ \cdots \circ MergeRotate(k, i + 1)$$

We then have the following.

▶ **Theorem 21.** *For a given $i < n$ and $k < 2^i$, let S be a k-block i-suffix-null solution space. Then $ItrMergeRotate(k, i)$ transforms S into a block.*

Proof. We prove by induction that for every $0 \leq j \leq n-i$, the solution space S after applying the transformation $MergeRotate(2^{j-1}k, i + j)$, is a $(2^j k)$-block $(i + j)$-suffix-null. It follows after applying $MergeRotate(2^{j-1}k, i + j)$ for $j = n - i$ that S is an ℓ-block n-suffix-null (for some $\ell \leq 2^{n-i}k$), i.e. S is a block.

In the base case $j = 0$ (i.e. before applying the first $MergeRotate$), S is by hypothesis a k-block i-suffix-null solution space. Assume by induction that, for some $0 \leq j \leq n - i - 1$ when applying $ItrMergeRotate(k, i)$ on S, after $MergeRotate(2^{j-1}k, i + j)$ we have that S is a $(2^j k)$-block $(i + j)$-suffix-null solution space. Then by definition, there is some m for which the subspace $S[\sigma_n, \dots, \sigma_{i+j+1}]$ is a null block for every $(\sigma_n, \dots, \sigma_{i+j+1}) >_{lex} bin(m)$, a block of size $2^j k$ for every $(\sigma_n, \dots, \sigma_{i+j+1}) <_{lex} bin(m)$ and $S[bin(m)]$ is a k_m block for some $k_m \leq 2^j k$.

Now the transformation $MergeRotate(2^j k, i+j+1)$ is applied on S, as described in Definition 17, by merging and rotating the subspaces $S[\sigma_n, \dots \sigma_{i+j+2}, 0]$ and $S[\sigma_n, \dots \sigma_{i+j+2}, 1]$ for every $(\sigma_n, \dots \sigma_{i+j+2}) \in \{0,1\}^{n-i-j-1}$. This makes four cases to consider:

1. $(\sigma_n, \dots \sigma_{i+j+2}, 1) < bin(m)$. Then both $S[\sigma_n, \dots \sigma_{i+j+2}, 0]$ and $S[\sigma_n, \dots \sigma_{n-i+j+2}, 1]$ are blocks of size $2^j k$, and therefore by Claim 18, applying $MergeRotate(2^j k, i+j+1)$ transforms $S[\sigma_n, \dots \sigma_{i+j+2}]$ to a block of size $2^{j+1} k$.
2. $(\sigma_n, \dots \sigma_{i+j+2}, 0) > bin(m)$. Then both $S[\sigma_n, \dots \sigma_{i+j+2}, 0]$ and $S[\sigma_n, \dots \sigma_{n-i+j+2}, 1]$ are null, and therefore applying $MergeRotate(2^j k, i+j+1)$ maintains $S[\sigma_n, \dots \sigma_{i+j+2}]$ null as well.
3. $(\sigma_n, \dots \sigma_{i+j+2}, 1) = bin(m)$. Then the subspace $S[\sigma_n, \dots \sigma_{i+j+2}, 0]$ is a block of size $2^j k$, while the subspace $S[\sigma_n, \dots \sigma_{i+j+2}, 1]$ is a block of size $k_m < 2^j k$. Then again by Claim 18, applying $MergeRotate(2^j k, i+j+1)$ transforms $S[\sigma_n, \dots \sigma_{i+j+2}]$ to a block of size $2^j k + k_m$, where $2^j k + k_m \leq 2^{j+1} k$.
4. $(\sigma_n, \dots \sigma_{i+j+2}, 0) = bin(m)$. Then the subspace $S[\sigma_n, \dots \sigma_{i+j+2}, 0]$ is a block of size k_m, while the subspace $S[\sigma_n, \dots \sigma_{i+j+2}, 1]$ is a null block. Since $k_m \leq 2^j k$ this case fits to the conditions of Lemma 19. Therefore we get that $S[\sigma_n, \cdots \sigma_{i+j+2}]$ is (still) a block of size $k_m \leq 2^{j+1} k$.

That shows that applying $MergeRotate(2^j k, i+j+1)$ on S in the j'th iteration of $ItrMergeRotate(k, i)$ transforms S to a $2^{j+1}k$-block $(i+j+1)$-suffix-null solution space as required. ◀

In the next section we see how to make use of Theorem 21 when blocking specific conjuncted Boolean formulas.

5.4 Transforming conjuncted variable-disjoint formulas

Having defined the iterated merge-rotate method, we finally demonstrate how to combine it with existing transformations to blocks, in the specific formulation which we now describe.

▶ **Theorem 22.** *Let φ be a Boolean formula such that $\varphi = \varphi_1 \wedge \varphi_2$ where φ_1 and φ_2 have disjoint variables. Furthermore, assume that T_1 and T_2 are Σ_1^P transformations that transform φ_1 and φ_2 respectively to blocks. Then there is a Σ_1^P transformation of size polynomial in $|T_1| + |T_2| + |ItrMergeRotate|$ that transforms φ to a block.*

Proof. We denote the φ_1 variables by $x_n \dots x_{i+1}$ for some i and the φ_2 variables by $x_i \dots, x_1$. We set an order on $x_n, \dots x_1$ where x_n is the msb. Denote the number of solutions in φ_1 by k_1 and the number of solutions of φ_2 by k_2 (we can obtain the k_i's by performing the transformations to blocks on φ_i's and use binary search.). Note that every solution subspace $S[\sigma_n, \dots \sigma_{i+1}]$ of $S[x_n, \dots x_{i+1}]$ is either null (when $\varphi_1(\sigma_n, \dots, \sigma_{i+1}) = 0$) or is an identical copy of the solution space of φ_2 (when $\varphi_1(\sigma_n, \dots, \sigma_{i+1}) = 1$).

We first apply $T_2' = T_2 \wedge \bigwedge_{j>i}(x_j \leftrightarrow y_j)$ on φ. This effectively applies T_2 to every copy of the solution space of φ_2 and so transforms every subspace of $S[x_n, \dots x_{i+1}]$ that is not null to a block of size k_2. Next, we apply $T_1' = T_1 \wedge \bigwedge_{j \leq i}(x_j \leftrightarrow y_j)$ to $T_2(\varphi)$. This transforms the solution space S to a k_2-block i-suffix-null. Finally we make use of Theorem 21 and apply $ItrMergeRotate(k_2, i)$ to transform S to a block. Thus the resulting composition $ItrMergeRotate(k_2, i) \circ T_1' \circ T_2'$ is a transformation that transforms φ to a block. Moreover, this composition is in Σ_1^P since all components are in Σ_1^P. ◀

Theorem 22 shows that there are cases in which a divide-and-conquer approach, in the sense of a syntactical decomposition of a Boolean formula into separate conjuncts, followed by pursuing a transformation for each sub formula separately to a block, can lead to an efficient transformation for the original formula to a block. This approach also follows recent methods in decomposition of Boolean formulas, see for example [10].

6 Conclusion

The transformations that we explored in this work transform Boolean functions, described by Boolean formulas, while maintaining the number of solutions. Manipulations of the structure of the solution space through Boolean formulas were done before on various occasions. One classical example in the theoretical setting is Sipser's proof of $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$ [3] that makes use of what we call the XOR transformation. In a more applied setting, the SAT community uses transformations in an ad-hoc manner as preprocessing steps; although some preprocessing techniques change the number of solutions, many techniques do not [4]. In addition, in the area of approximate model counting, two very recent works [2, 7] suggest the use of transformations to create a high degree of separation (in terms of Hamming distance) between solutions in the solution space in order to improve practical approximate counting. It would be interesting in future work to see how our work on transformations can be applied with this goal. Finally, it is also worth mentioning a similar line of work that studies the Formula Isomorphism Problem, which asks if there exists a bijection between variables such that two Boolean formulas become equivalent [1, 6, 18].

To the best of our knowledge, this work is the first that formally defines and studies the general concept of transformations of Boolean functions described as Boolean formulas. Among the results that we presented here are not only takeaways on the computational complexity limitations of using transformations, but also definitions, properties, and foundational techniques that express the versatility and the usability in which transformations can be used to combinatorially manipulate various solution spaces.

References

1 Manindra Agrawal and Thomas Thierauf. The formula isomorphism problem. *SIAM Journal on Computing*, 30(3):990–1009, 2000.

2 S. Akshay and Kuldeep S. Meel. Scalable Approximate Model Counting via Concentrated Hashing. Under submission.

3 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

4 Armin Biere. Preprocessing and Inprocessing Techniques in SAT. In *Proceedings of HVC*, page 1, 2011. doi:10.1007/978-3-642-34188-5_1.

5 Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS Press, 2009.

6 Elmar Böhler, Edith Hemaspaandra, Steffen Reith, and Heribert Vollmer. Equivalence and isomorphism for Boolean constraint satisfaction. In *International Workshop on Computer Science Logic*, pages 412–426. Springer, 2002.

7 Michele Boreale and Daniele Gorla. Approximate Model Counting, Sparse XOR Constraints and Minimum Distance, 2019. arXiv:1907.05121.

8 Randal E Bryant. Graph-based algorithms for boolean function manipulation. Technical report, Cargnegie-Mellon University, 2001.

9 Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. From weighted to unweighted model counting. In *Proceedings of IJCAI*, 2015.

10 Supratik Chakraborty, Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. Functional Synthesis via Input-Output Separation. In *Proceedings of FMCAD*, pages 1–9, 2018.

11 Yves Crama and Peter L Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.

12 Carmel Domshlak and Jörg Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *J. of AI Research*, 30:565–620, 2007.

13 Lance Fortnow. Counting complexity. *Complexity theory retrospective II*, pages 81–107, 1997.

14 Dror Fried, Axel Legay, Joël Ouaknine, and Moshe Y. Vardi. Sequential Relational Decomposition. In *Proceedings of LICS*, pages 432–441, 2018.

15 Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, 1953.

16 Yehuda Naveh, Michal Rimon, Itai Jaeger, Yoav Katz, Michael Vinov, Eitan Marcu, and Gil Shurek. Constraint-based random stimuli generation for hardware verification. *AI Magazine*, 28(3):13–13, 2007.

17 Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In *Proceedings of IJCAI*, pages 3141–3148, 2015.

18 B.V. Rao and M.N. Sarma. Isomorphism testing of read-once functions and polynomials. In *Proceedings of FSTTCS*, 2011.

19 Tian Sang, Fahiem Bacchus, Paul Beame, Henry A Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *Proceedings of SAT*, pages 20–28, 2004.

20 Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of STOC*, pages 118–126. ACM, 1983.

21 Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM J. on Computing*, 8(3):410–421, 1979.

22 Ingo Wegener. *The complexity of Boolean functions*. BG Teubner, 1987.

Two-Way Parikh Automata

Emmanuel Filiot
Université libre de Bruxelles, Belgium

Shibashis Guha
Université libre de Bruxelles, Belgium

Nicolas Mazzocchi
Université libre de Bruxelles, Belgium

Abstract

Parikh automata extend automata with counters whose values can only be tested at the end of the computation, with respect to membership into a semi-linear set. Parikh automata have found several applications, for instance in transducer theory, as they enjoy a decidable emptiness problem.

In this paper, we study two-way Parikh automata. We show that emptiness becomes undecidable in the non-deterministic case. However, it is PSpace-C when the number of visits to any input position is bounded and the semi-linear set is given as an existential Presburger formula. We also give tight complexity bounds for the inclusion, equivalence and universality problems. Finally, we characterise precisely the complexity of those problems when the semi-linear constraint is given by an arbitrary Presburger formula.

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

Keywords and phrases Parikh automata, two-way automata, Presburger arithmetic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.40

Related Version `https://arxiv.org/abs/1907.09362`

Funding *Emmanuel Filiot*: Research associate of F.R.S.-FNRS. He was supported by the ARC Project Transform Fédération Wallonie-Bruxelles, the FNRS CDR project J013116F and "Synapse" FNRS MIS project J013116F.
Shibashis Guha: Supported by the ARC project "Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond" (Fédération Wallonie-Bruxelles).
Nicolas Mazzocchi: PhD student funded by a FRIA fellowship from the F.R.S.-FNRS.

1 Introduction

Parikh automata, introduced in [18], extend finite automata with counters in $\mathbb{Z}$ which can be incremented and decremented, but the counters can only be tested at the end of the computation, for membership in a semi-linear set (represented for instance as an existential Presburger formula). More precisely, transitions are of the form $(q, \sigma, \vec{v}, q')$ where q, q' are states, σ is an input symbol and $\vec{v} \in \mathbb{Z}^d$ is a vector of dimension d. A word w is accepted if there exists a run ρ on w reaching an accepting state and whose final vector (the component-wise sum of all vectors along ρ) belongs to a given semi-linear set. Parikh automata strictly extend the expressive power of finite automata. For example, the context-free language of words of the form $a^n b^n$ is definable by a deterministic Parikh automaton which checks membership in $a^* b^*$, counts the number of occurrences of a and b, and at the end tests for equality of the counters, i.e. membership in the linear set $\{(n, n) \mid n \in \mathbb{N}\}$. They still enjoy decidable, NP-C, non-emptiness problem [9].

Parikh automata (PA) have found applications for instance in *transducer* theory, in particular to the equivalence problem of functional transducers on words, and to check structural properties of transducers [10], as well as in answering queries in graph databases [9].

Extensions of Parikh automata with a pushdown stack have been considered in [17] with positive decidability results with respect to emptiness. Two-way Parikh automata with a visibly pushdown stack have been considered in [6] with applications to tree transducers.

In this paper, our objective is to study *two-way Parikh automata* (2PA), the extension of PA with a two-way input head, where the semi-linear set is given by an existential Presburger formula. For 2PA as well as subclasses such as deterministic 2PA (2DPA), we aim at characterizing the precise complexity of their decision problems (membership, emptiness, inclusion, equivalence), and analysing their expressiveness and closure properties.

Contributions. Since semi-linear sets are closed under all Boolean operations, it is easily seen that deterministic Parikh automata (DPA) are closed under all Boolean operations. More interestingly, it is also known that, while they strictly extend the expressive power of DPA, *unambiguous* PA (UPA) are (non-trivially) closed under complement (as well as union and intersection) [2]. We give here a simple explanation to these good closure properties: UPA *effectively* correspond to 2DPA. Closure of 2DPA under Boolean operations indeed holds straightforwardly due to determinism. The conversion of UPA to 2DPA is however non-trivial, but is obtained by the very same result on word transducers: it is known that unambiguous finite transducers are equivalent to two-way deterministic finite transducers [21], based on a construction by Aho, Hopcroft and Ullman [1], recently improved by one exponential in [7]. Parikh automata can be seen as transducers producing sequences of vectors (the vectors occurring on their transitions), hence yielding the result. The conversion of 2DPA to UPA is a standard construction based on *crossing sections*, which however needs to be carefully analysed for complexity purposes.

The effective equivalence between 2DPA and UPA indeed entails decidability of the non-emptiness problem for 2DPA. However, given that non-emptiness of PA is known to be NP-C [9], and the conversion of 2DPA to UPA is exponential, this leads to NExpTime complexity. By a careful analysis of this conversion and small witnesses properties of Presburger formulas, we show that emptiness of 2DPA, and even *bounded-visit* 2PA, is actually PSpace-C. Bounded-visit 2PA are non-deterministic 2PA such that for some natural number k, each position of an input word w is visited at most k times by any accepting computation on w. In particular, 2DPA are always n-visit for n the number of states. If the number k of visits is a fixed constant, non-emptiness is then NP-C, which is consistent with the complexity result of [9] for (one-way) PA (by taking $k = 1$). We show that dropping the bounded-visit restriction however leads to undecidability.

Thanks to the closure properties of 2DPA, we show that the inclusion, universality and equivalence problems are all coNExpTime-C. Those problems are known to be undecidable for PA [18]. The membership problem of 2PA turns out to be NP-C, just as for (one-way) PA. The coNExpTime lower bound holds for one-way deterministic Parikh automata, a result which is also new, to the best of our knowledge.

Finally, we study the extension of two-way Parikh automata with a semi-linear set defined by a Σ_i-Presburger formula, i.e. a formula with a *fixed* number i of unbounded blocks of quantifiers where the consecutive blocks alternate $i-1$ times between existential and universal blocks, and the first block is existential. We characterise tightly the complexity of the non-emptiness problem for bounded-visit Σ_i-2PA, as well as the universality, inclusion and equivalence problems for Σ_i-2DPA, in the weak exponential hierarchy [13]. For $i > 1$, we find that the complexity of these problems is dominated by the complexity of checking satisfiability or validity of Σ_i-Presburger formulas. This is unlike the case $i = 1$: the non-emptiness problem for bounded-visit 2PA is PSpace-C while satisfiability of Σ_1-formulas is NP-C.

Related work. Parikh automata are known to be equivalent to reversal-bounded multicounter machines (RBCM) [16] in the sense that they describe the same class of languages [2]. Two-way RBCM (2RBCM), even deterministic, are known to have undecidable emptiness problem [16]. Using diophantine equations as in [16], we show that emptiness of 2PA is undecidable. However our decidability result for 2DPA contrasts with the undecidabilty of deterministic 2RBCM emptiness. The difference is that 2RBCM can test their counters at any moment during a computation, and not only at the end. Based on the fact that the number of reversals is bounded, deferring the tests at the end of the computation is always possible [16] but non-determinism is needed. Unlike 2DPA, deterministic 2RBCM are not necessarily bounded-visit. A 2DPA can be seen as a deterministic 2RBCM whose tests on counters are only done at the end of a computation.

Two-way Parikh automata on *nested words* have been studied in [6] where it is shown that under the *single-use* restriction (a generalisation of the bounded-visit restriction to nested words), they have NExpTime-C non-emptiness problem. Bounded-visit 2PA are a particular case of those Parikh automata operating on (non-nested) words. Applying the result of [6] to 2PA would yield a non-optimal NExpTime complexity for the non-emptiness problem, as it first goes through an explicit but exponential transformation into a one-way machine with known NP-C non-emptiness problem. Here instead, we rely on a small witness property, whose proof uses a transformation into one-way Parikh automaton, and then we apply a PSpace algorithm performing on-the-fly the one-way transformation up to some bounded length.

Finally, the emptiness problem for the intersection of n PA was shown to be PSpace-C in [9]. Our PSpace-C result on 2PA emptiness generalises this result, as the intersection of n PA can be simulated trivially by a (sweeping) n-bounded 2PA. The main lines of our proof are similar to those in [9], but in addition, it needs a one-way transformation on top of the proof in [9], and a careful analysis of its complexity.

2 Two-way Parikh automata

Two-way Parikh automata are two-way automata extended with weight vectors and a semi-linear acceptance condition. In this section, we first define two-way automata, semi-linear sets and then two-way Parikh automata.

Two-way Automata. A *two-way finite automaton* (2FA for short) A over an alphabet Σ is a tuple $(Q, Q^{\mathsf{L}}, Q^{\mathsf{R}}, Q_I, Q_H, Q_F, \Delta)$ whose components are defined as follows. We let $\vdash$ and $\dashv$ be two delimiters not in Σ, intended to represent the beginning and the end of the word respectively. The set Q is a non-empty finite set of states partitioned into the set of right-reading states Q^{R} and the set of left-reading states Q^{L}. Then, $Q_I \subseteq Q^{\mathsf{R}}$ is the set of initial states, $Q_H \subseteq Q$ is the set of halting states, and $Q_F \subseteq Q_H$ is the set of accepting states. The states belonging to $Q_H \setminus Q_F$ are said to be *rejecting*. Finally, $\Delta \subseteq Q \times (\Sigma \cup \{\vdash, \dashv\}) \times Q$ is the set of transitions. Intuitively, the reading head of A is always placed in between input positions, a transition from $q \in Q^{\mathsf{R}}$ (resp. $q \in Q^{\mathsf{L}}$) reads the input letter on the right (resp. left) of the head and moves the head one step to the right (resp. left). Also, we have the following restrictions on the behaviour of the head to keep it in between the boundaries $\vdash$ and $\dashv$ and to ensure the following properties on the initial and the halting states:

1. no outgoing transition from a halting state:
   $(Q_H \times (\Sigma \cup \{\vdash, \dashv\}) \times Q) \cap \Delta = \varnothing$

2. the head cannot move left (resp. right) when it is to the left of $\vdash$ (resp. right of $\dashv$): $(Q^{\mathsf{L}} \times \{\vdash\} \times Q^{\mathsf{L}}) \cap \Delta = \varnothing$ (resp. $(Q^{\mathsf{R}} \times \{\dashv\} \times (Q^{\mathsf{R}} \setminus Q_F)) \cap \Delta = \varnothing$)
3. all transitions leading to a halting state q_H read the delimiter $\dashv$: $((q, a, q_H) \in \Delta \wedge q_H \in Q_H) \implies (q \in Q^{\mathsf{R}} \wedge a = \dashv)$

A configuration $(u^{\mathsf{L}}, p, u^{\mathsf{R}})$ of A on a word $u \in \Sigma^*$ consists of a state p and two words $u^{\mathsf{L}}, u^{\mathsf{R}} \in (\Sigma \cup \{\vdash, \dashv\})^*$ such that $u^{\mathsf{L}}u^{\mathsf{R}} = \vdash u \dashv$. A *run* ρ on a word $u \in \Sigma^*$ is a sequence $\rho = (u_0^{\mathsf{L}}, q_0, u_0^{\mathsf{R}})a_1(u_1^{\mathsf{L}}, q_1, u_1^{\mathsf{R}}) \ldots a_n(u_n^{\mathsf{L}}, q_n, u_n^{\mathsf{R}})$ alternating between configurations on u and letters in $\Sigma \cup \{\vdash, \dashv\}$ such that for all $1 \leq i \leq n$, we have $(q_{i-1}, a_i, q_i) \in \Delta$, and for all $s \in \{\mathsf{L}, \mathsf{R}\}$, if $q_{i-1} \in Q^s$ then $|u_i^s| = |u_{i-1}^s| - 1$. The length of the run ρ, denoted $|\rho|$ is the number of letters appearing in ρ. Here $|\rho| = n$. The run ρ is *halting* if $q_n \in Q_H$ (and hence $u_n^{\mathsf{R}} = \varepsilon$ by condition 3), *initial* if $u_0^{\mathsf{L}} = \varepsilon$ and $q_0 \in Q_I$, *accepting* if it is both initial and halting, and $q_n \in Q_F$; otherwise the run is *rejecting.* A word u is accepted by A if there exists an accepting run of A on u, and the language $L(A)$ of A is defined as the set of words it accepts.

An automaton A is said to be *one-way* (FA) if Q^{L} is empty. A run ρ is said to be *k-visit* if every input position is visited at most k times in the run ρ, i.e. for $\rho = (u_0^{\mathsf{L}}, q_0, u_0^{\mathsf{R}}) \ldots (u_n^{\mathsf{L}}, q_n, u_n^{\mathsf{R}})$, we have $\max\{|P| \mid P \subseteq \{0, \ldots, n\} \wedge \forall i, j \in P,\ u_i^{\mathsf{L}} = u_j^{\mathsf{L}}\} \leq k$. The automaton A is said to be k-visit if all its accepting runs are k-visit, *fixed-visit* if it is k-visit for some fixed k and *bounded-visit* if it is k-visit for some unfixed k. Also, A is said to be *deterministic* if for all $p \in Q$ and all $a \in \Sigma \cup \{\vdash, \dashv\}$ there exists at most one $q \in Q$ such that $(p, a, q) \in \Delta$. Finally, it is *unambiguous* (denoted by the class 2UFA or UFA depending on whether it is two-way or one-way) if for every input word there exists at most one accepting run. The following proposition is trivial but useful:

▶ **Proposition 2.1.** *Any bounded-visit 2FA with n states is k-visit for some $k \leq n$.*

Semi-linear Sets. Let $d \in \mathbb{N}_{\neq 0}$. A set $L \subseteq \mathbb{Z}^d$ of dimension d is *linear* if there exist $\vec{v}_0, \ldots, \vec{v}_k \in \mathbb{Z}^d$ such that $L = \{\vec{v}_0 + \sum_{i=1}^k x_i\vec{v}_i \mid x_1, \ldots, x_n \in \mathbb{N}\}$. The vectors $(\vec{v}_i)_{1\leq i\leq k}$ are the *periods* and $\vec{v}_0$ is called the *base*, forming what we call a *period-base representation* of L, whose size is $d \cdot (k+1) \cdot \log_2(\mu+1)$ where μ is the maximal absolute integer appearing on the vectors. A set is *semi-linear* if it is a finite union of linear sets. A period-base representation of a semi-linear set is given by a period-base representation for each of the linear sets it is composed of, and its size is the sum of the sizes of all those representations.

Alternatively, a semi-linear set of dimension d can be represented as the set of models of a Presburger formula with d free variables. A Presburger formula is a first-order formula built over terms t on the signature $\{0, 1, +, \times_2\} \cup X$, where X is a countable set of variables and $\times_2$ denotes the doubling (unary) function[1]. In particular, Presburger formulas obey the following syntax:

$$\Phi \overset{\text{def}}{=} t \leq t \mid \exists x\ \Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \neg\Phi$$

The class of formulas of the form $\exists\vec{x}_1, \forall\vec{x}_2 \ldots, \Omega_i\vec{x}_i\,[\varphi]$ where φ is quantifier free and $\Omega \in \{\forall, \exists\}$ is denoted by $\boldsymbol{\Sigma}_i$. In particular, $\boldsymbol{\Sigma}_1$ is the set of existential Presburger formulas. The size $|\Psi|$ of a formula is its number of symbols. We denote by $\vec{v} \models \varphi$ the fact that a vector $\vec{v}$ of dimension d satisfies a formula φ with d free variables, and say that φ is satisfiable if there exists such a $\vec{v}$. The formula φ is said to be valid if it is satisfied by any $\vec{v}$. It is well-known [12] that a set $S \subseteq \mathbb{Z}^d$ is semi-linear iff there exists an existential Presburger formula ψ with d free variables such that $S = \{\vec{v} \mid \vec{v} \models \psi\}$.

[1] The function $\times_2$ is syntactic sugar allowing us to have simpler binary encoding of values.

Let $\Sigma = \{a_1, \dots, a_n\}$ be an alphabet (assumed to be ordered), and $u \in \Sigma^*$, the Parikh image of u is defined as the vector $\mathfrak{P}(u) = (|u|_{a_1}, \dots, |u|_{a_n})$ where $|u|_a$ denotes the number of times a occurs in u. The Parikh image of language $L \subseteq \Sigma^*$ is $\mathfrak{P}(L) = \{\mathfrak{P}(u) | u \in L\}$. Parikh's theorem states that the Parikh image of any context-free language is semi-linear.

Two-way Parikh automata. A two-way Parikh automaton (2PA) of dimension $d \in \mathbb{N}$ over Σ is a tuple $P = (A, \lambda, \psi)$ where $A = (Q, Q^{\mathsf{L}}, Q^{\mathsf{R}}, Q_I, Q_H, Q_F, \Delta)$ is a 2FA over Σ, $\lambda \colon \Delta \to \mathbb{Z}^d$ maps transitions to vectors, and ψ is an *existential* Presburger formula with d free variables, and is called the *acceptance constraint.* The *value* $V(\rho)$ of a run ρ of A is the sum of the vectors occurring on its transitions, with $V(\rho) = 0_{\mathbb{Z}^d}$ if $|\rho| = 0$. A word is accepted by P if it is accepted by some accepting run ρ of A and $V(\rho) \models \psi$. The language $L(P)$ of P is the set of words it accepts. The automaton P is said to be *one-way*, *two-way*, *k-visit*, *unambiguous* and *deterministic* if its underlying automaton A is so. We define the representation size[2] of P as $|P| = |Q| + |\psi| + |\text{range}(\lambda)|\big(d \log_2(\mu + 1) + |Q|^2\big)$ where $\text{range}(\lambda) = \{\lambda(t) \mid t \in \Delta\}$ and μ is the maximal absolute entries appearing in weight vectors of P. Finally two 2PA are *equivalent* if they accept the same language.

Examples. Let $\Sigma = \{a, b, c, \#\}$ and for all $n \in \mathbb{N}$, let $L_n = \{a^k\#u \mid u \in \{b, c\}^* \wedge k = |\{i \mid 1 \leq i \leq |u| - n \wedge u[i] \neq u[i + n]\}|\}$, i.e. k is the number of positions i in u such that the ith letter $u[i]$ mismatches with $u[i + n]$. For all n, L_n is accepted by the 2DPA of Fig. 1 which has $O(n)$ states, tagged with R or L to indicate whether they are right- or left-reading respectively. On a word w, the automaton starts by reading a^k and increments its counter to store the value k (state q^a). Then, for the first $|u| - n$ positions i of u, the automaton checks whether $u[i] \neq u[i + n]$ in which case the counter is decremented. To do so, it stores $\sigma = u[i]$ in its state, moves $n + 1$ times to the right (states $q_0, q_1^\sigma, \dots, q_n^\sigma$), checks whether $u[i + n] \neq u[i]$ (transitions q_n^σ to p_1) and decrements the counter accordingly. Then, it moves n times to the left (states p_1 to p_n). Whenever it reads $\dashv$ from states q_j^σ, p_j or q_0, it moves to state q_F and accepts if the counter is zero.

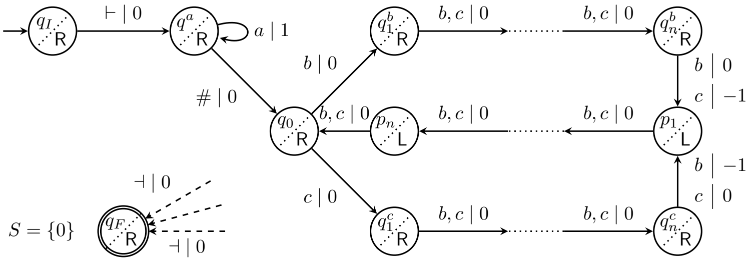


Figure 1 A 2DPA recognising $L_n = \{a^k\#u \mid u \in \{b, c\}^* \wedge k = |\{i \mid 1 \leq i \leq |u| - n \wedge u[i] \neq u[i + n]\}|\}$.

Our second example shows how to encode multiplication. The language $\{a^n\#a^m\#a^{n\times m} \mid n, m \in \mathbb{N}\}$ is indeed definable by the 2PA of Figure 2 which has dimension 2. When reading a word of the form $a^n\#a^m\#a^\ell$, every accepting run makes k passes over a^n where k is chosen non-deterministically by the choice made on state q_1 on reading $\#$. Along those k passes, the automaton increments the first dimension whenever a is read in a right-to-left pass. It

[2] Note that weight vectors are not memorized on transitions but into a table and transitions only carry a key of this table to refer the corresponding weight vectors.

also counts the number of passes in the second dimension. Thus, when entering state q_2, the sum of the vectors so far is (nk, k). Then, on a^m, it decrements the second dimension and on a^ℓ, it decrements the first dimension, and eventually checks that both the counters are equal to zero, which implies that $k = m$ and $\ell = nk = nm$. Note that this automaton is not bounded-visit as its number of visits to any position of a^n is arbitrary.

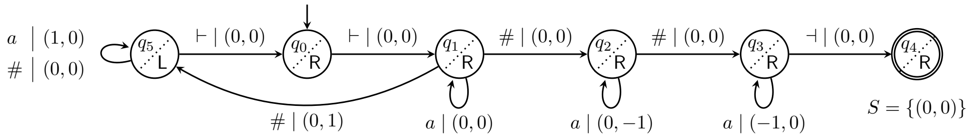


Figure 2 A 2PA recognising $\{a^n \# a^m \# a^{n\times m} \mid n, m \in \mathbb{N}\}$.

3 Relating two-way and one-way Parikh automata

In this section, we provide an algorithm which converts a bounded-visit 2PA into a PA defining the same language, through a *crossing section* construction. This technique is folkloric in the literature (see Section 2.6 of [15]) and has been introduced to convert a 2FA into an equivalent FA. Intuitively, the one-way automaton is constructed such that on each position i of the input word, it guesses a tuple of transitions (called crossing section), triggered by the original two-way automaton at the same position i and additionally checks a local validity between consecutive tuples (called *matching* property). A one-way automaton takes crossing sections as set of states. Furthermore, the matching property is defined to ensure that the sequence of crossing sections which successively satisfy it, correspond to the sequence of crossing sections of an accepting two-way run. Thanks to the commutativity of $+$, the order in which weights are combined by the two-way automaton does not matter and therefore, transitions of the one-way automaton are labelled by summing the weights of transitions of the crossing section. Formally, we define a crossing section as follows:

▶ **Definition 3.1** (crossing section). *Let $k \in \mathbb{N}_{\neq 0}$. Consider a k-visit 2PA $P = (A, \lambda, \psi)$ over Σ and $a \in \Sigma \cup \{\vdash, \dashv\}$. An a-crossing section is a sequence $c = (p_1, a, q_1) \dots (p_\ell, a, q_\ell) \in \Delta^+$ such that $1 \leq \ell \leq k$, $p_1, q_\ell \in Q^{\mathsf{R}}$ and for all $m \in \{\mathsf{L}, \mathsf{R}\}$, $p_i \in Q^m \implies p_{i+1} \notin Q^m$. We define the value of c as $V(c) = \sum_{i=1}^{\ell} \lambda(p_i, a, q_i)$, and its length $|c| = \ell$. The L-anchorage of c is defined by $p_1 f(q_2, p_3) \dots f(q_{\ell-1}, p_\ell)$ where $f(q_i, p_{i+1}) = \varepsilon$ if $q_i = p_{i+1}$ and $q_i \in Q^{\mathsf{R}}$, otherwise $f(q_i, p_{i+1}) = q_i p_{i+1}$. The R-anchorage of c is defined by $f(q_1 p_2) \dots f(q_{\ell-2} p_{\ell-1}) q_\ell$ where $f(q_i, p_{i+1}) = \varepsilon$ if $q_i = p_{i+1}$ and $q_i \in Q^{\mathsf{L}}$, otherwise $f(q_i, p_{i+1})$ is the identity. Furthermore, c is said to be* initial *if its L-anchorage is $p_1 \in Q_I$. Dually, c is said to be* accepting *if its R-anchorage is $q_\ell \in Q_F$.*

Given a run ρ of a 2PA over u and a position $1 \leq i \leq |u|$, the *crossing section of ρ at position i* is defined as the sequence of all transitions triggered by ρ when reading the ith letter, taken in the order of appearance in ρ. We also define the *crossing section sequence* $\mathcal{C}(\rho)$ as the sequence of crossing sections of ρ from position 1 to $|u|$. Note that the first crossing section is initial and the last crossing section of ρ is accepting if ρ is accepting.

▶ **Example 3.2.** Figure 3, shows a run over the word $\vdash ab \dashv$. Consider the a-crossing section $c = (p_1, a, q_1)(p_2, a, q_2)(p_2, a, q_3)(p_4, a, q_4)(p_5, a, q_5)$ with $q_1 = p_2, q_2 = p_3$ and $q_4 = p_5$. In particular the run makes on immediate reversal at those states, and exits the a-crossing section from q_3 to q_5. The L-anchorage of c is $p_1 f(q_2, p_3) f(q_4, p_5) = p_1$, the R-anchorage of c is $f(q_1, p_2) f(q_3, p_4) q_5 = q_3 p_4 q_5$ and $V(c) = \vec{v}_2 + \vec{v}_3 + \vec{v}_4 + \vec{v}_{11} + \vec{v}_{12}$. Note that the states of the crossing section do not appear in the anchorage when the run changes its reading direction.

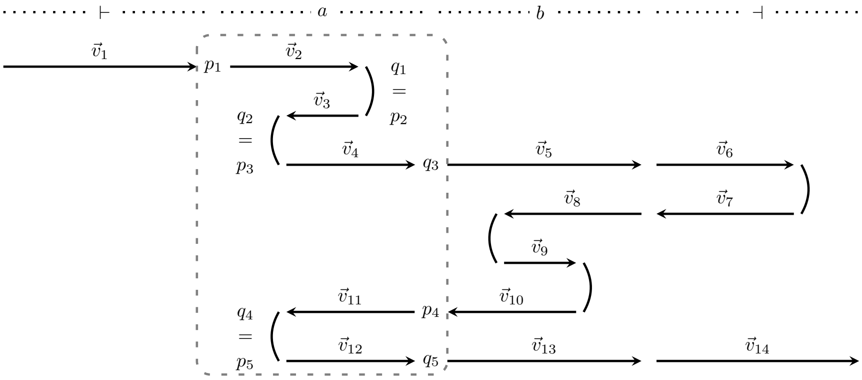


Figure 3 A a-crossing section of a run.

▶ **Definition 3.3** (matching relation). *Consider two crossing sections c_1, c_2 from the same automaton. The matching relation M is defined such that $(c_1, c_2) \in M$ if the* R*-anchorage of c_1 equals the* L*-anchorage of c_2.*

In general, an arbitrary sequence of crossing sections may not correspond to a run of a two-way automaton, that is a crossing section sequence $s = c_1, \dots, c_\ell$ such that $\mathcal{C}(\rho) \neq s$ for all run ρ. Lemma 3.4 shows that the matching property ensures the existence of such a run ρ in the two-way automaton.

▶ **Lemma 3.4.** *Consider $s = c_1, \dots, c_n$ where c_i is an a_i-crossing section such that c_1 is initial, c_n is accepting, and $(c_i, c_{i+1}) \in M$ for all $i \in \{1, \dots, n-1\}$. Then there exists an accepting two-way run ρ over $a_1 \dots a_n$ such that $\mathcal{C}(\rho) = s$. Moreover, $V(\rho) = \sum_{i=1}^{n} V(c_i)$.*

▶ **Theorem 3.5.** *Let $k \in \mathbb{N}_{\neq 0}$. Given a k-visit 2PA P, one can effectively construct an equivalent PA R that is at most exponentially bigger. Furthermore, if P is deterministic then R is unambiguous.*

Proof. Let $P = (A, \lambda, \psi)$ with $A = (Q, Q^{\mathsf{L}}, Q^{\mathsf{R}}, Q_I, Q_H, Q_F, \Delta)$ be a k-visit 2PA of dimension d with $n = |Q|$ states. In this proof we show how to construct $R = (B, \omega, \psi)$ where $B = (V, V^{\mathsf{L}}, V^{\mathsf{R}}, V_I, V_H, V_F, \Gamma)$ is a PA of dimension d having $\mathcal{O}(n^{2k})$ states such that $|\text{range}(\omega)| \leq |\text{range}(\lambda)|^{k+1}$. Note that the formula ψ is the same in both P and R.

To do so, we first consider a symbol $\top$ and extend the relation M such that $(c, \top) \in M$ holds for all accepting crossing section c. Then, we define R as follows:

- V is the set of crossing sections of length at most k
- V_I is the set of initial crossing sections and $V_H = V_F = \{\top\}$
- $\Gamma = \{(c_1, a, c_2) \in V \times (\Sigma \cup \{\vdash, \dashv\}) \times V \mid (c_1, c_2) \in M \wedge c_1$ is an a-crossing section$\}$
- $\omega\colon (c_1, a, c_2) \mapsto V(c_1)$

Similar to the case of 2FA, a word u is accepted by B if there exists an accepting run of B on u, and the language $L(B)$ of B is defined as the set of words it accepts. The inclusion $L(R) \subseteq L(P)$ is a direct consequence of Lemma 3.4, while the other direction is based on the following observation: any accepting two-way run ρ has a sequence of crossing sections $\mathcal{C}(\rho)$, consecutively satisfying the matching relation. Note that, the choice of c_2 in a transition (c_1, a, c_2) is non-deterministic in general; but when P is deterministic at most one such choice of c_2 will correspond to a two-way run ensuring unambiguity. ◀

The previous crossing section construction permits to construct a one-way automaton from a bounded-visit two-way automaton. This construction is exponential in the number of states and in the number of distinct weight vectors. Nevertheless, a close inspection of the proof of Theorem 3.5, reveals that the exponential explosion in the number of distinct weight vectors can be avoided, while preserving the non-emptiness (but not the language).

▶ **Lemma 3.6.** *Let P be a k-visit 2PA. We can effectively construct a PA R with $\mathcal{O}(n^{2k})$ states and such that $L(R) = \varnothing$ iff $L(P) = \varnothing$. Furthermore, R has the same set of weight vectors and the same acceptance constraint as P.*

Proof. The construction is the same as in Theorem 3.5 but each transition of the one-way automaton $t = (c_1, a, c_2)$ is split into the following $|c_1|$ consecutive transitions, using a fresh symbol $\# \notin \Sigma$: $c_1 \xrightarrow{a} (t, 1) \xrightarrow{\#} (t, 2) \xrightarrow{\#} \dots (t, |c_1| - 2) \xrightarrow{\#} (t, |c_1| - 1) \xrightarrow{\#} c_2$. The vectors of those transitions are defined as follows. If $c_1[i]$ denotes the ith transition of c_1, then the vector of the first R-transition is the vector of the P-transition $c_1[1]$, and the vector of any R-transition from state (t, i) is the vector of the P-transition $c_1[i + 1]$. The two languages are then equal modulo erasing # symbols. ◀

▶ **Theorem 3.7.** *Unambiguous Parikh automata have the same expressiveness as two-way deterministic (even reversible*[3]*) Parikh automata i.e. UPA = 2DPA. Furthermore, the transformation from one formalism to the other can be done in* ExpTime.

Proof. We only show here UPA ⊆ 2DPA. The opposite direction is given by Theorem 3.5. Let $P = (A, \lambda, \psi)$ be a UPA of dimension d over Σ. Consider the alphabet $\Lambda \subseteq \mathbb{Z}^d$ as the set of vectors occurring on the transitions of P. We can see the automaton A with the morphism λ as an unambiguous finite transducer T defining a function from Σ^* to Λ^*. It is known that any unambiguous letter-to-letter one-way transducer can be transformed into an equivalent letter-to-letter deterministic two-way transducer. This result is explicitly stated in Theorem 1 of [21] which is based on a general technique introduced by Aho, Hopcroft and Ullman [1][4]. Recently, another approach has been introduced which reduces the complexity of the previous technique by one exponential [7], and allows to show that any unambiguous finite transducer is equivalent to a reversible two-way transducer exponentially bigger, yielding our result. ◀

4 Emptiness Problem

The emptiness problem asks, given a 2PA, whether the language it accepts is empty. We have seen in Example 2 how to encode the multiplication of two natural numbers encoded in unary. We can generalise this to the encoding of solutions of Diophantine equations as languages of 2PA, yielding undecidability:

▶ **Theorem 4.1.** *The emptiness problem for 2PA is undecidable.*

The proof of this theorem relies on the fact that an input position can be visited an arbitrary number of times, due to non-determinism. If instead we forbid this, we recover decidability. To prove it, we proceed in two steps: first, we rely on the result of the previous

[3] An automaton is said to be reversible if it is both deterministic and co-deterministic.

[4] Based on the technique of Aho and Hopcroft and Ullman a similar result was shown in [4] for weighted automata, namely that an unambiguous weighted automata over a semiring can be converted into an equivalent deterministic two-way weighted automata.

section showing that any bounded-visit 2PA can be effectively transformed into some (one-way) PA. This yields decidability of the emptiness problem as this problem is known to be decidable for PA. To get a tight complexity in PSPACE, we analyse this transformation (which is exponential), to get exponential bounds on the size of shortest non-emptiness witnesses. A key lemma is the following, whose proof gathers ideas and arguments that already appeared in [20, 9].

▶ **Lemma 4.2.** *Let P be a one-way Parikh automaton with n states and γ distinct weight vectors. Then, we can construct an existential Presburger formula $\varphi(x) = \bigvee_{i=1}^{m} \varphi_i(x)$ such that for all $\ell \in \mathbb{N}$, $\varphi(\ell)$ holds iff there exists $w \in L(P) \cap \Sigma^\ell$. Furthermore, $\log_2(m)$ and each φ_i are* poly$(|P|, \log n)$*, in addition φ can be constructed in time $2^{\mathcal{O}(\gamma^2 \log(\gamma n))}$.*

▶ Remark 4.3. Note that, $\varphi(x)$ is not in *prenex normal form* (PNF) but φ_i are. Since φ is a disjunction of PNF subformulas, it can be in PNF in polynomial time.

Thanks to the lemma above, we are able to show that the non-emptiness problem for bounded-visit 2PA is PSPACE-C, just as the non-emptiness problem for two-way automata. In some sense, adding semi-linear constraints to two-way automata is for free as long as it is bounded-visit.

▶ **Theorem 4.4.** *The non-emptiness problem for bounded-visit 2PA is* PSPACE-C*. It is* NP-C *for k-visit 2PA when k is fixed.*

Proof. Consider a k-visit 2PA $P = (A, \lambda, \psi)$ of dimension d. We start with the PSPACE membership. Intuitively, we first want to apply Lemma 3.6 in order to deal with a one-way automaton, and apply then Lemma 4.2 to reduce the non-emptiness problem of the one-way Parikh automaton to the satisfiability of an existential Presburger formula. Nevertheless, we cannot explicitly transform P into a one-way automaton while keeping polynomial space. So, in the sequel, (i) we highlight an upper bound on the smallest witness of non-emptiness and based on it, (ii) we provide an NPSPACE algorithm which decides if there exists such a witness.

(i) By Lemma 4.2 applied on the PA obtained from Lemma 3.6, there exists an existential Presburger formula $\varphi(\ell) = \bigvee_{i=1}^{m} \varphi_i(\ell)$ where each $|\varphi_i|$ is polynomial in $|P|$. This formula is satisfiable iff there exists $w \in \Sigma^\ell$ such that $w \in L(P)$. By Theorem 6 (A) of [22], there exists N exponential in $|\varphi_i|$ such that φ_i is satisfiable iff $\varphi_i(\ell)$ holds for some $0 \leq \ell \leq N$. Hence, there exists N exponential in $|P|$ such that $\min\{|u| \mid u \in L(P)\} \leq N$.

(ii) The algorithm guesses a witness u of length at most N on-the-fly and a run on it. It controls its length by using a binary counter: as N is exponential in $|P|$, the memory needed for that counter is polynomial in $|P|$. The transitions of the one-way automaton obtained from Lemma 3.6 can also be computed on-demand in polynomial space. Eventually, it suffices to check that the last state is accepting and the sum $\vec{v} = (v_1, \dots, v_d)$ of the vectors computed on-the-fly along the run satisfies the Presburger formula $\psi(x_1, \dots, x_d)$. To do so, our algorithm constructs a closed formula $\psi^{\vec{v}}$ in polynomial time such that $\psi^{\vec{v}}$ is true iff $\vec{v} \models \psi$. It is possible by hardcoding the values of $\vec{v}$ in ψ by substituting each x_i by a term t_{v_i} of size $(\log_2(v_i))^2$ encoding v_i, by using the function symbol $\times_2$ e.g. $t_{13} = \times_2(\times_2(\times_2(1))) + \times_2(\times_2(1)) + 1$. Let us argue that $\psi^{\vec{v}}$ has polynomial size. Let μ be the maximal absolute entry of vectors of P, then $v_i \leq \mu N$, and since N is exponential in $|P|$, t_{v_i} has polynomial size in $|P|$ and $\log_2(\mu)$. Hence $\psi^{\vec{v}}$ has polynomial size, and its satisfiability can be checked in NP [22].

The lower bound is direct as it already holds for the emptiness problem of deterministic two-way automata, by a trivial encoding of the PSPACE-C intersection problem of n DFA [19].

When k is fixed, then the conversion to a one-way automaton (Lemma 3.6) is polynomial. Then, the result follows from the NP-C result for the non-emptiness of PA [9]. ◀

▶ Remark 4.5. In [9], non-emptiness is shown to be polynomial time for PA when the dimension is fixed, the values in the vectors are unary encoded and the semi-linear constraint is period-base represented. As a consequence, for all fixed d, k, the non-emptiness problem for k-visit 2PA with vectors in $\{0,1\}^d$ and a period-base represented semi-linear constraint can be solved in PTime.

5 Closure properties, universality, inclusion and equivalence problems

Since the class of 2DPA is equivalent to the class of UPA that is known to be closed under Boolean operations [3, 18], we get the closure properties of 2DPA for free, although with non-optimal complexity. We show here that they can be realised in linear-time for intersection and union. For the complement however, while the size of the state-space stays linear, the size of the acceptance condition explodes due to the transformation of negated existential Presburger formulas into existential formulas.

▶ **Theorem 5.1** (Boolean closure). *Let P, P_1, P_2 be 2DPA such that $P = (A, \lambda, \psi)$. One can construct a 2DPA $\overline{P} = (A', \lambda', \psi')$ such that $L(\overline{P}) = \overline{L(P)}$ and the size of A' is linear in the size of A. One can construct in linear-time a 2DPA $P_\cup$ (resp. $P_\cap$) such that $L(P_\cup) = L(P_1) \cup L(P_2)$ (resp. $L(P_\cap) = L(P_1) \cap L(P_2)$).*

Proof. Let us start by intersection, assuming $P_i = (A_i, \lambda_i, \psi_i)$ has dimension d_i. The automaton $P_\cap$ is constructed with dimension $d_1 + d_2$. Then $P_\cap$ first simulates P_1 on the first d_1 dimensions (with weight vectors belonging to $\mathbb{Z}^{d_1} \times \{0\}^{d_2}$), and then, if P_1 eventually reaches a halting state, it stops if it is non-accepting and rejects, otherwise it simulates P_2 on the last d_2 dimensions with vectors in $\{0\}^{d_1} \times \mathbb{Z}^{d_2}$, and accepts the word if the word is accepted by P_2 as well. The Presburger acceptance condition is defined as $\psi(\vec{x}_1, \vec{x}_2) = \psi_1(\vec{x}_1) \wedge \psi_2(\vec{x}_2)$. Note that if P_1 never reaches a halting state, then $P_\cap$ won't either, so the word is rejected by both automata. It is also a reason why this construction cannot be used to show closure under union: even if P_1 never reaches a halting state, it could well be the case that P_2 accepts the word, but the simulation of P_2 in that case will never be done. However, assuming that P_1 halts on any input, closure under union works with a similar construction. Additionally, we need to keep in some new counter c the information whether P_1 has reached an accepting state: First $P_\cup$ simulates P_1, if P_1 halts in some accepting state, then c is incremented and $P_\cup$ proceeds with the simulation of P_2. The formula is then $\psi(\vec{x}_1, \vec{x}_2, c) = (c = 1 \wedge \psi_1(\vec{x}_1)) \vee \psi_2(\vec{x}_2)$.

So, we have closure under union in linear-time as long as P_1 halts on every input. This can be used to show closure under complement, using the following observation: $\overline{L(P)} = \overline{L(A)} \cup L(A, \lambda, \neg\psi)$ and moreover, it is known that 2DFA can be complemented in linear-time into a 2DFA which always halts [11]. The formula $\neg\psi$ is universal since ψ is existential. Then, $\neg\psi$ could be converted into an equivalent existential formula using quantifier elimination [5] of doubly exponential size.

For the closure under union, we use the equality $L(P_1) \cup L(P_2) = \overline{\overline{L(P_1)} \cap \overline{L(P_2)}}$. It can be done in linear-time because the formulas for $\overline{P_1}$ and $\overline{P_2}$ are universal, and so is the formula for the 2DPA accepting $\overline{L(P_1)} \cap \overline{L(P_2)}$. By applying again the complement construction, we get an existential formula (without using quantifier eliminations). ◀

Thanks to Theorem 5.1 and decidability of non-emptiness for 2DPA, we easily get the decidability of the universality problem (deciding whether $L(P) = \Sigma^*$), the inclusion problem (deciding whether $L(P_1) \subseteq L(P_2)$), and the equivalence problem (deciding whether

$L(P_1) = L(P_2)$) for 2DPA. The following theorem establishes tight complexity bounds. It is a consequence of a more general result (Theorem 6.4) that we establish for Parikh automata with *arbitrary* Presburger formulas in Section 6.

▶ **Theorem 5.2.** *The universality, inclusion and equivalence problems are* coNExpTime-C *for 2DPA.*

Finally, we study the membership problem which asks given a Parikh automaton P and a word $w \in \Sigma^*$, whether $w \in L(P)$. Hardness was known already for PA [9].

▶ **Theorem 5.3.** *The membership problem for 2PA is* NP-C*.*

6 Parikh automata with arbitrary Presburger acceptance condition

In this section, we consider Parikh automata where the acceptance constraint is given as an arbitrary Presburger formula, that is, not restricted to existential Presburger formula, and we study the complexity of their decision problems. For all $i > 0$, a two-way Σ_i-Parikh automaton (Σ_i-2PA for short) is a tuple $P = (A, \lambda, \Psi)$ where A, λ are defined just as for 2PA and $\Psi \in \Sigma_i$. In particular, a Σ_1-2PA is exactly a 2PA. Similarly, we also define Σ_i-DPA, Σ_i-2DPA, Σ_i-PA as expected, and their Π_i counterpart (when the formula is in Π_i).

The complexity of Presburger arithmetic has been connected to the weak ExpTime hierarchy [14, 13] which resides between NExpTime and ExpSpace. It is defined as $\bigcup_{i\geq 0} \Sigma_i^{\text{Exp}}$ where:

$$\begin{array}{lll} \Sigma_0^{\mathrm{P}} \overset{\text{def}}{=} \Pi_0^{\mathrm{P}} \overset{\text{def}}{=} \mathrm{PTime} & \Sigma_{i+1}^{\mathrm{P}} \overset{\text{def}}{=} \mathrm{NP}^{\Sigma_i^{\mathrm{P}}} & \Pi_{i+1}^{\mathrm{P}} \overset{\text{def}}{=} \mathrm{coNP}^{\Sigma_i^{\mathrm{P}}} \\ \Sigma_0^{\mathrm{Exp}} \overset{\text{def}}{=} \Pi_0^{\mathrm{Exp}} \overset{\text{def}}{=} \mathrm{ExpTime} & \Sigma_{i+1}^{\mathrm{Exp}} \overset{\text{def}}{=} \mathrm{NExpTime}^{\Sigma_i^{\mathrm{P}}} & \Pi_{i+1}^{\mathrm{Exp}} \overset{\text{def}}{=} \mathrm{coNExpTime}^{\Sigma_i^{\mathrm{P}}} \end{array}$$

Since Lemma 4.2 uses the acceptance constraint as a black box, we can generalise it as follows.

▶ **Lemma 6.1.** *For any fixed $i \in \mathbb{N}_{\neq 0}$, given a Σ_i-PA P with n states and γ distinct weight vectors, we can construct a Σ_i-formula Φ such that for all $\ell \in \mathbb{N}$ we have that $\Phi(\ell) = \bigvee_{j=1}^{m} \Phi_j(\ell)$ holds iff there exists $w \in L(P) \cap \Sigma^{|\ell|}$. Furthermore, $\log_2(m)$ and the size of each Φ_j are $\mathsf{poly}(|P|, \log(n))$, in addition Φ can be constructed in time $2^{\mathcal{O}(\gamma^2 \log(\gamma n))}$.*

Using Lemma 6.1, we can extend Theorem 4.4 to bounded-visit Σ_{i+1}-2PA. Note that the case of Σ_1-2PA is not covered by the following statement.

▶ **Theorem 6.2.** *For any fixed $i \in \mathbb{N}_{\neq 0}$, the non-emptiness problem for bounded-visit Σ_{i+1}-2PA is Σ_i^{Exp}-C.*

Proof. For the upper-bound, we show that this problem can be solved by an alternating Turing machine in exponential time, which alternates at most i times between sequences of non-deterministic and universal transitions, starting with non-deterministic transitions (called i-alternating machine in the sequel). As shown in [13], the satisfiability of Σ_{i+1}-formulas is complete for Σ_i^{Exp}-C. Hence there is an i-alternating machine $\mathcal{M}$ running in exponential time which checks the satisfiability of such formulas. Now, similar to the case of Σ_1 in Theorem 4.4, from a bounded-visit Σ_{i+1}-2PA P one can construct a Σ_{i+1}-formula which is true iff the automaton has a non-empty language. We can do so by applying Lemma 6.1 on the PA obtained[5] from Lemma 3.6. Hence, non-emptiness of a bounded-visit Σ_{i+1}-2PA

[5] Lemma 3.6 can be trivially adapted to Σ_i-formulas as acceptance condition.

reduces to satisfiability of a Σ_{i+1}-formula $\Phi(\ell) = \bigvee_{j=1}^{m} \Phi_j(\ell)$ such that $\log_2(m)$ and the size of each Φ_j are polynomial in $|P|$ and can be constructed in time $2^{\mathcal{O}(\gamma^2 \log(\gamma n))}$. However we cannot construct explicitly Φ, since its size is exponential in $|P|$. Instead we construct an i-alternating machine $\mathcal{M}'$ that first guesses a disjunct Φ_s and constructs it in exponential time, and then simulates the machine $\mathcal{M}$ on Φ_s. Recall the $\mathcal{M}$ starts with non-deterministic transitions. Thus the machine $\mathcal{M}'$ runs in exponential time, and also performs only i alternations, which provides Σ_i^{Exp} upper bound.

Hardness comes from checking if a Σ_{i+1}-sentence holds true, which is Σ_i^{Exp}-C as shown in [13]. From a Σ_{i+1}-sentence Ψ it suffices to construct a Parikh automaton $P = (A, \lambda, \Psi)$ of dimension 0 such that $L(A) \neq \varnothing$, therefore $L(P) \neq \varnothing$ iff $L(P) = L(A)$ iff Ψ holds. ◀

▶ **Theorem 6.3** (Boolean closure). *Let P, P_1, P_2 be Σ_i-2DPA. One can construct in linear time a Π_i-2DPA $\overline{P}$ and two Σ_i-2DPA $P_\cup, P_\cap$ such that $L(\overline{P}) = \overline{L(P)}$, $L(P_\cup) = L(P_1) \cup L(P_2)$ and $L(P_\cap) = L(P_1) \cap L(P_2)$.*

Proof. The constructions are the same as in the proof of the case $i = 1$ of Theorem 5.1, using closure under disjunction and conjunction of Σ_i and the fact that negating a Σ_i-formula yields a Π_i-formula. ◀

▶ **Theorem 6.4.** *For all fixed $i \in \mathbb{N}_{\neq 0}$, the universality, inclusion and equivalence problems for Σ_i-2DPA are Π_i^{Exp}-C.*

Proof. We first prove the upper bound for the most general problem which is inclusion. Let $P_i = (A_i, \lambda_i, \psi_i)$ be a Σ_i-2DPA. Note that $L(P_1) \subseteq L(P_2)$ iff $L(P_1) \cap \overline{L(P_2)} = \varnothing$. So, using Theorem 6.3 we first construct in linear-time a Π_i-2DPA $\overline{P_2} = (A_2', \lambda_2', \Psi_2')$ such that $L(\overline{P_2}) = \overline{L(P_2)}$ and then $P_\cap = (A, \lambda, \Psi)$ such that $L(P_\cap) = L(P_1) \cap L(\overline{P_2})$. From the construction in Theorem 5.1 generalised to Σ_i-2DPA, recall that the formula Ψ is defined as $\Psi(\vec{x}_1, \vec{x}_2) = \Psi_1(\vec{x}_1) \wedge \Psi_2'(\vec{x}_2)$. Let $\Psi_1(\vec{x}_1) = \exists\vec{y}_1\forall\vec{y}_2 \dots \Omega\vec{y}_i\,[\varphi_1(\vec{x}_1, \vec{y}_1, \dots, \vec{y}_i)]$, and $\Psi_2'(\vec{x}_2) = \forall\vec{z}_1\exists\vec{z}_2 \dots \mho\vec{z}_i\,[\varphi_2(\vec{x}_2, \vec{z}_1, \dots, \vec{z}_i)]$ where $\Omega, \mho \in \{\exists, \forall\}$ such that $\Omega \neq \mho$. Hence Ψ is equivalent to the following Σ_{i+1}-formula.

$$\exists\vec{y}_1\forall\vec{z}_1\forall\vec{y}_2\exists\vec{z}_2\exists\vec{y}_3 \dots \Omega\vec{z}_{i-1}\vec{y}_i\mho\vec{z}_i\Big[\varphi_1(\vec{x}_1, \vec{y}_1, \dots, \vec{y}_i) \wedge \varphi_2(\vec{x}_2, \vec{z}_1, \dots, \vec{z}_i)\Big]$$

Finally, emptiness of $P_\cap$ can be decided in Π_i^{Exp} by Theorem 6.2.

For the lower bound, we show that the universality problem of Σ_i-DPA is Π_i^{Exp}-hard. This holds even for a fixed number of states and vector values in $\{-1, 0, 1\}$, showing that the complexity comes from the formula part. From a Σ_i-formula Ψ with d free variables, we construct a Parikh automaton $P = (A, \lambda, \Psi)$ of dimension d over alphabet $\Sigma = \{a_i^+, a_i^-\}_{1 \leq i \leq d}$. Any word w over Σ defines a valuation $\mu_w(x_i) = |w|_{a_i^+} - |w|_{a_i^-}$ for all $1 \leq i \leq d$. Conversely, any valuation μ can be encoded as a word over Σ. Hence, Ψ holds for all values iff for all $w \in \Sigma^*$, we have $\mu_w \models \Psi$. We construct a deterministic one-way automaton A such that $L(A) = \Sigma^*$ and for all $w \in \Sigma^*$, the value of the run r over w is μ_w. The automaton A has one accepting and initial state q over which it loops and, when reading a_i^+ (resp. a_i^-) it increases dimension i by 1 (resp. by -1). ◀

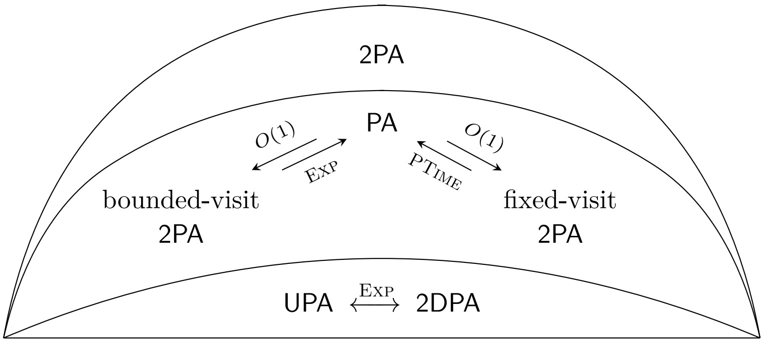
▶ Remark 6.5. Since a 2DPA is a Σ_1-2DPA, and the class coNExpTime is the same as Π_1^{Exp}, we have that Theorem 6.4 for $i = 1$ is exactly the same as Theorem 5.2.

7 Conclusion

In this paper, we have provided tight complexity bounds for the emptiness, inclusion, universality and equivalence problems for various classes of two-way Parikh automata. We have shown that when the semi-linear constraint is given as a Σ_i-formula, for $i > 1$, the complexity of those problems is dominated by the complexity of checking satisfiability or validity of Σ_i-formulas. We have shown that 2DPA (resp. bounded-visit 2PA) have the same expressive power as unambiguous (one-way) PA (resp. non-deterministic PA). Remark that the same techniques apply to show that 2UPA are equivalent to 2DPA, and hence to UPA, exactly as it is done for string transducer in [7, 8].

In terms of succinctness, it is already known that 2DFA are exponentially more succinct than FA, witnessed for instance by the family $D_n = \{uu \mid u \in \{0,1\}^* \wedge |u| = n\}$. However D_n is accepted by a PA with polynomially many states in n and vectors of dimension $2n$ which permit to store each input letters and check equality with the acceptance constraint. We conjecture that 2DPA are exponentially more succinct than PA, witnessed by the language L_n of Section 2. We leave as future work the introduction of techniques allowing to prove such results (pumping lemmas), as the dimension and acceptance constraint size has to be taken into account as well, as shown with D_n.

Finally, we plan to extend the pattern logic of [10], which intensively uses (one-way) Parikh automata for its model-checking algorithm, to reason about structural properties of two-way machines, and use two-way Parikh automata emptiness checking algorithms for model-checking this new logic.



Two-way automata	Non-emptiness	Universality & Inclusion
2PA	undecidable	undecidable
bounded-visit 2PA	PSpace-C	undecidable
fixed-visit 2PA	NP-C	undecidable
2DPA	NP-C	coNExpTime-C
bounded-visit Σ_i-2PA	$\Sigma_{i-1}^{\mathrm{Exp}}$-C	undecidable
fixed-visit Σ_i-2PA	$\Sigma_{i-1}^{\mathrm{Exp}}$-C	undecidable
Σ_i-2DPA	$\Sigma_{i-1}^{\mathrm{Exp}}$-C	Π_i^{Exp}-C

Figure 4 Summary of expressivenesses and complexities where bounded-visit 2PA (resp. fixed-visit 2PA) holds for k-visit 2PA for some k (resp. for some fixed k).

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. A general theory of translation. *Mathematical Systems Theory*, 3(3):193–221, 1969.

2 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the Expressiveness of Parikh Automata and Related Models. In *NCMA'11 Proceedings*, pages 103–119, 2011.

3 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous Constrained Automata. *Int. J. Found. Comput. Sci.*, 24(7):1099–1116, 2013.

4 Vincent Carnino and Sylvain Lombardy. On Determinism and Unambiguity of Weighted Two-Way Automata. *IJFCS*, 26(8):1127–1146, 2015.

5 David C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7(1):91––99, 1972.

6 Luc Dartois, Emmanuel Filiot, and Jean-Marc Talbot. Two-Way Parikh Automata with a Visibly Pushdown Stack. In *FoSSaCS'19 Proceedings*, pages 189–206, 2019.

7 Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. On Reversible Transducers. In *ICALP'18 Proceedings*, pages 113:1–113:12, 2017.

8 Rodrigo de Souza. Uniformisation of Two-Way Transducers. In *LATA'13 Proceedings*, pages 547–558, 2013.

9 Diego Figueira and Leonid Libkin. Path Logics for Querying Graphs: Combining Expressiveness and Efficiency. In *LICS'15 Proceedings*, pages 329–340, 2015.

10 Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. A Pattern Logic for Automata with Outputs. In *DLT'18 Proceedings*, pages 304–317, 2018.

11 Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173–1187, 2007.

12 Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

13 Christoph Haase. Subclasses of presburger arithmetic and the weak Exp hierarchy. In *LICS'14 Proceedings*, pages 47:1–47:10, 2014.

14 Lane A. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39:299–322, 1987.

15 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

16 Oscar H. Ibarra. Reversal-Bounded Multicounter Machines and Their Decision Problems. *Journal ACM*, 25(1):116–133, 1978.

17 Wong Karianto. Parikh Automata with Pushdown Stack, 2004.

18 Felix Klaedtke and Harald Rueß. Monadic Second-order Logics with Cardinalities. In *ICALP'03 Proceedings*, pages 681–696, 2003.

19 Dexter Kozen. Lower bounds for natural proof systems. *Foundations of Computer Science*, pages 254–266, 1977.

20 Anthony Widjaja Lin. *Model checking infinite-state systems: generic and specific approaches*. PhD thesis, University of Edinburg, 2010.

21 Michal P. Chytil and Vojtech Jákl. Serial Composition of 2-Way Finite-State Transducers and Simple Programs on Strings. In *ICALP'77 Proceedings*, pages 135–147, 1977.

22 Bruno Scarpellini. Complexity of subcases of presburger arithmetic. *American Mathematical Society*, 284(1):203–218, 1984.

The Well Structured Problem for Presburger Counter Machines

Alain Finkel

LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, IUF, France
alain.finkel@ens-paris-saclay.fr

Ekanshdeep Gupta

Chennai Mathematical Institute, Chennai, India
ekanshdeep@cmi.ac.in

Abstract

We introduce the *well structured problem* as the question of whether a model (here a counter machine) is well structured (here for the usual ordering on integers). We show that it is undecidable for most of the (Presburger-defined) counter machines except for Affine VASS of dimension one. However, the *strong well structured problem* is decidable for *all* Presburger counter machines. While Affine VASS of dimension one are not, in general, well structured, we give an algorithm that computes the set of predecessors of a configuration; as a consequence this allows to decide the well structured problem for 1-Affine VASS.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Verification by model checking

Keywords and phrases Well structured transition systems, infinite state systems, Presburger counter machines, reachability, coverability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.41

Related Version `https://arxiv.org/abs/1910.02736`

Funding The work reported was carried out in the framework of ReLaX, UMI2000 (ENS Paris-Saclay, CNRS, Univ. Bordeaux, CMI, IMSc). This work was also supported by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS).

1 Introduction

Context. Well Structured Transition Systems (WSTS) [9, 8] are a well-known model to solve termination, boundedness, control-state reachability and coverability problems. It is well known that Petri nets and Vector Addition Systems with States (VASS) are WSTS and that Minsky machines are not WSTS. But the characterization of counter machines which are well structured (resp. with strong monotony) is surprisingly unknown. Moreover, given a counter machine, can we decide whether it is well structured (resp. with strong monotony)? These questions are relevant since a positive answer could allow to verify *particular instances* of undecidable models like Minsky machines and counter machines. In this paper, we consider *Presburger counter machines (PCM)* where each transition between two control-states is labelled by a Presburger formula which describes how each counter is modified by the firing of the transition. The PCM model includes Petri nets, Minsky machines and most of the counter machine models studied in the literature, for example counter machines where transitions between control-states are given by affine functions having Presburger domains [3, 11].

Affine VASS. In an Affine VASS (AVASS), transitions between control-states are labelled by affine functions whose matrices have elements in $\mathbb{Z}$ (and not in $\mathbb{N}$ as usual). AVASS extends VASS (where transitions are translations) and positive affine VASS (introduced as

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 41; pp. 41:1–41:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

self-modified nets in [21] and studied as affine well structured nets in [13]. [4] extends the Rackoff technique to AVASS where all matrices are larger than the identity matrix: for this subclass, coverability and boundedness are shown in EXPSPACE. The variation of VASS which may go below 0, called $\mathbb{Z}$-VASS, is studied in [15] and for their extension, $\mathbb{Z}$-*Affine* VASS, reachability is shown NP-complete for VASS with resets, PSPACE-complete for VASS with transfers and undecidable in general [2, 1]; let us remark that all $\mathbb{Z}$-Affine VASS have *positive* matrices.

Moreover AVASS allow the simulation of the zero-test so they are at least as expressive as Minsky machines. But for dimension one, AVASS are more expressive than Minsky machines: in fact, $Post^*$ is computable as a Presburger formula for 1-counter Minsky machines but this is not the case for 1-AVASS which can generate the set of all the powers of 2 (this set is not the solution of any Presburger formula).

The computation of the set Pre^* of all predecessors of a configuration is effective for 2-VASS (extended with one zero-test and resets) [12] as a Presburger formula and for pushdown automata [5] as a regular language. But the computation of Pre^* fails for 3-VASS and for Pushdown VAS since Pre^* is neither semilinear nor regular [19].

Our contributions. We introduce two new problems related to well structured systems and Presburger counter machines. The so-called *well structured problem*: (1) given a PCM, is it a WSTS? and the *strong well structured problem*: (2) given a PCM, is it a WSTS with *strong* monotony?

We prove that the well structured problem is undecidable for PCM even if restricted to dimension one (1-PCM) with just Presburger functions (i.e. piecewise affine functions); undecidability is also verified for Affine VASS in dimension two (2-Affine VASS). The undecidability proofs use the fact that Minsky machines can be simulated by both 1-PCM and 2-Affine VASS. However, we prove the decidability of the well structured problem for 1-Affine VASS (which subsumes 1-Minsky machines). Since the strong monotony can be expressed as a Presburger formula, the strong well structured problem is decidable for *all* PCMs. These results are summarised below:

	Well Structured Problem	Strong Well Structured Problem
PCM	U	**D**
Functional 1-PCM	**U** [Theorem 14]	D
2-AVASS	U	D
2-Minsky machines	**U** [Theorem 15]	D
1-AVASS	**D** [Theorem 26]	D

We give an algorithm that computes Pre^* of a 1-AVASS and this extends a similar known result for 1-Minsky machines and 1-VASS (and for pushdown automata [5]). The computation of Pre^* allows us to give a simple proof that reachability and coverability are decidable for 1-AVASS (in fact reachability is known to be PSPACE-complete for polynomial one-register machines [10] which contains 1-AVASS). Moreover, the computation of Pre^* allows to decide the well structured problem for 1-AVASS. These results are summarised below:

	Reachability	Coverability
1-PCM (functional)	U	**U** [Corollary 19]
1-AVASS	**D** [Corollary 24]	D
d-totally positive AVASS	**D** [Theorem 29]	D
d-positive AVASS ($d \geq 2$)	**U** [Theorem 28]	**D** [WSTS]
2-AVASS	U	**U** [Corollary 18]

Outline. We introduce in Section 2 two models, well structured transition systems (WSTS) and Presburger counter machines (PCM); we show that the property for an ordering to be well is undecidable. Section 3 analyses the decidability of the well structured problems for many classes of PCM and Affine VASS. Section 4 studies the decidability of reachability and coverability for the classes studied in Section 3.

Due to space constraints, some proofs are deferred to an extended version of this paper freely available online under the same title.

2 Counter machines and WSTS

A relation $\leq$ on a set E is a *quasi ordering* if it is reflexive and transitive; it is an ordering if moreover $\leq$ is antisymetric. A quasi ordering $\leq$ on E is a *well quasi ordering (wqo)* if for all infinite sequences of elements of E, $(e_i)_{i \in \mathbb{N}}$, there exists two indices $i < j$ such that $e_i \leq e_j$. For an ordered set $(E, \leq)$ and a subset $X \subseteq E$, the upward closure of X denoted by $\uparrow X$ is defined as follows: $\uparrow X = \{x \mid \exists y \in X \text{ such that } y \leq x\}$. X is said to be *upward closed* if $X = \uparrow X$.

2.1 Arithmetic counter machines

A *d-dim arithmetic counter machine (short, d-arithmetic counter machine or an arithmetic counter machine)* is a tuple $M = (Q, \Phi, \rightarrow)$ where Q is a finite set of control-states, Φ is a set of logical formulae with $2d$ free variables $x_1, ..., x_d, x'_1, ..., x'_d$ and $\rightarrow \subseteq Q \times \Phi \times Q$ is the transition relation between control-states. We can also without loss of generality assume that $\rightarrow$ covers Φ, i.e. Φ does not have unnecessary formulae. A configuration of M refers to an element of $Q \times \mathbb{N}^d$. The operational semantics of a d-arithmetic counter machine M is a transition system $S_M = (Q \times \mathbb{N}^d, \rightarrow)$ where $\rightarrow \subseteq (Q \times \mathbb{N}^d) \times (Q \times \mathbb{N}^d)$ is the transition relation between configurations. For a transition (q, ϕ, q') in M, we have a transition $(q; x_1, ..., x_d) \rightarrow (q'; x'_1, ..., x'_d)$ in S_M iff $\phi(x_1, ..., x_d, x'_1, ..., x'_d)$ holds. Note that we are slightly abusing notation by using the same $\rightarrow$ for both M and S_M. We may omit Φ from the definition of a counter machine if it is clear from context.

A d-dim arithmetic counter machine M with *initial configuration* c_0 is defined by the tuple $M = (Q, \Phi, \rightarrow, c_0)$ where $(Q, \Phi, \rightarrow)$ is a d-arithmetic counter machine and $c_0 \in Q \times \mathbb{N}^d$ is the initial configuration. An arithmetic counter machine is *effective* if the transition relation is decidable (there is a decidable procedure to determine if there is a transition $x \rightarrow y$ between any two configurations x, y) and this is the case when it is given by an algorithm, a recursive relation, or decidable first order formulae (for instance Presburger formulae). An arithmetic counter machine is said to be *functional* if each formula in Φ that labels a transition in M defines a partial function.

Most usual counter machines can be expressed with Presburger formulae. It is well known that Presburger arithmetic with congruence relations without quantifiers is equivalent in expressive power to standard Presburger arithmetic [14].

▶ **Definition 1.** *A* Presburger counter machine (PCM) *is an arithmetic counter machine $M = (Q, \Phi, \rightarrow)$ such that Φ is a set of Presburger formulae with congruence relations without quantifiers.*

▶ **Proposition 2** ([6]). *The property for a d-dim PCM to be* functional *is decidable in NP.*

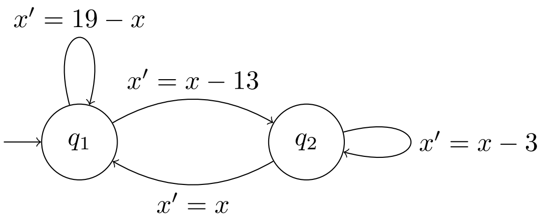


Figure 1 The counter machine M_1.

Minsky machines with d counters are d-PCM $M = (Q, \Phi, \to)$ where Φ consists of either translations with upwards closed guards, or formulae of the form $\wedge_{i=1}^{d}(x_i = x'_i) \wedge x_k = 0$ for varying k (zero-tests). *Vector Addition Systems with States (VASS)* are Minsky machines without zero-tests. An *Affine VASS* with d counters (d-AVASS) is a d-PCM where each transition is labelled by a formula equivalent to an affine function of the form $f(x) = Ax + b$ where $A \in M_d(\mathbb{Z})$ is a $d \times d$ matrix over $\mathbb{Z}$ and $b \in \mathbb{Z}^d$. The domain of such a function would be the (Presburger) set of all $x \in \mathbb{N}^d$ such that $Ax + b \in \mathbb{N}^d$. For convenience, we will denote d-AVASS transitions by a pair $(A, b) \in M_d(\mathbb{Z}) \times \mathbb{Z}^d$. Note that AVASS is an extension of VASS where transitions are not labelled by vectors but by affine functions (A_i, b_i). Let us define positive and totally-positive AVASS. A *positive* AVASS S is an AVASS such that every matrix A_i of S is positive. This model has been studied for instance in [13]. A *totally-positive* AVASS S is a positive AVASS such that every vector b_i of S is positive. For totally positive AVASS, an instance of the boundedness problem has been shown decidable in [13]. Note that we say something is positive if it is greater than or equal to 0, not strictly greater than 0.

▶ **Example 3.** The machine M_1 in Figure 1 is a 1-AVASS but it is not a 1-VASS because there is a negative transition from q_1 to q_1.

▶ **Proposition 4** ([6]). *Checking whether a given PCM is a VASS, AVASS, positive AVASS or a totally positive AVASS is decidable.*

2.2 Well structured transition systems

A *transition system* is a tuple $S = (X, \to)$ where X is a (potentially infinite) set of configurations and $\to \subseteq X \times X$ is the transition relation between configurations. We denote by $\xrightarrow{*}$ the reflexive and transitive closure of $\to$. For a subset $S \subseteq X$, we denote by $Pre(S) := \{t \mid t \to s \text{ for some } s \in S\}$, and $Pre^*(S) := \{t \mid t \xrightarrow{*} s \text{ for some } s \in S\}$. Similarly for $Post(S)$ and $Post^*(S)$.

An *ordered transition system* $S = (X, \to, \leq)$ is a transition system $(X, \to)$ with a quasi-ordering $\leq$ on X. Given two configurations $x, y \in X$, x is said to *cover* y if there exists a configuration $y' \geq y$ such that $x \xrightarrow{*} y'$. An ordered transition system $S = (X, \to, \leq)$ is *monotone*, if for all configurations $s, t, s' \in X$ such that $s \to t$, $s' \geq s$ implies that s' covers t. S is *strongly monotone* if for all configurations $s, t, s' \in X$ such that $s \to t$, $s' \geq s$ implies that there exists $t' \geq t$ such that $s' \to t'$.

▶ **Definition 5** ([8]). *A* well structured transition system *(WSTS) is an ordered transition system $S = (X, \to, \leq)$ such that $(X, \leq)$ is a wqo and S is monotone.*

The *coverability problem* is to determine, given two configurations s and t, whether there exists a configuration t' such that $s \xrightarrow{*} t' \geq t$ (s covers t). This problem is one often studied alongside well-structuredness.

Let us consider the usual wqo $\leq$ on $Q \times \mathbb{N}^d$ associated with a d-counter machine $M = (Q, \rightarrow)$: $(q_1; x_1, x_2, ..., x_d) \leq (q_2; y_1, ..., y_d) \iff (q_1 = q_2) \wedge (\wedge_{i=1}^{d} x_i \leq y_i)$.

We say that an arithmetic counter machine $M = (Q, \Phi, \rightarrow)$ is *well structured* (or is a WSTS) iff its associated transition system S_M is a WSTS under the usual ordering. Since the usual ordering on $(Q \times \mathbb{N}^d, \leq)$ is a wqo, let us remark that the associated ordered transition system $S_M = (Q \times \mathbb{N}^d, \rightarrow, \leq)$ is a WSTS iff S_M is monotone.

Given a counter machine $M = (Q, \rightarrow)$, the *control-state reachability problem* is that given a configuration $(q; n_1, ..., n_d)$, and a control-state q' whether there exist values of counters $(m_1, ..., m_d)$ such that $(q; n_1, ..., n_d) \xrightarrow{*} (q'; m_1, ..., m_d)$. In this case, we often say that q' is *reachable* from $(q; n_1, ..., n_d)$.

We introduce two new problems related to WSTS and Presburger counter machines.

- The *well structured problem*: given a PCM, is it a WSTS?
- The *strong well structured problem*: given a PCM, is it a WSTS with strong monotony?

▶ **Example 6.** The machine M_1 (Figure 1) is not strongly monotone since we have: $(q_1, 0) \xrightarrow{x'=19-x} (q_1, 19)$. However, we see that $Post^*(q_1, 10) = \{(q_1, 9), (q_1, 10)\}$. Therefore we can deduce that $(q_1, 10)$ cannot cover $(q_1, 19)$. Hence M_1 is not well structured. We give, in Section 4, an algorithm for deciding whether a 1-AVASS is well structured.

It is shown in [8] that *almost* every transition system can be turned into a WSTS for the *termination ordering* which is not, in general, decidable. So the problem is not only to decide whether a system is a WSTS in general; we have to choose a *decidable* ordering. We show that deciding whether arbitrary (non-effective) transition systems are well-structured for the usual (decidable) ordering on natural numbers is undecidable.

▶ **Proposition 7.** *The well structured problem for* 1*-arithmetic counter machines is undecidable.*

We now show that restricting to *effective* transition systems does not allow us to decide the property of being a WSTS.

▶ **Corollary 8.** *The well structured problem (for the usual ordering on* $\mathbb{N}$*) for effective transition systems whose set of configurations is included in* $\mathbb{N}$ *is undecidable.*

Proof. There exists a reduction from the Halting Problem as follows:

Given a Turing machine M, we define a transition system $S_M = (\mathbb{N}, \rightarrow_M)$ as follows: If $(m = 0) \vee (M$ does not halt in m steps), then, for all n, there is a transition $m \rightarrow_M n$. Hence this transition relation $\rightarrow_M$ is decidable. Now, if M does not halt, then there is a transition $m \rightarrow_M n$ for all $m, n \in \mathbb{N}$. This satisfies monotony, hence in this case, S_M is a WSTS. However, if M halts in exactly m steps, then there is no transition from $m + 1$ but there is, in any case, a transition from 0 to n for all n. Hence in this case, S_M is not a WSTS. Therefore, S_M is a WSTS iff T does not halt. ◀

2.3 Testing whether an ordering is well

In the previous results, the usual well ordering on natural numbers is not necessarily the unique *decidable* ordering when considering the well structured problem for counter machines. Let $\leq$ be a decidable quasi ordering relation on $\mathbb{N}^d$. If we are interested in whether a counter machine with this ordering is WSTS, it raises the natural question of whether we can decide if $\leq$ is a wqo. Unfortunately, but unsurprisingly, we first show that this property is undecidable in dimension one ($d = 1$).

▶ **Proposition 9.** *The property for a decidable ordering on $\mathbb{N}$ to be a well ordering is undecidable.*

Let us study the case of *Presburger-definable orderings* in $\mathbb{N}$. Among many equivalent characterizations of wqo, we know that a quasi ordering is well iff it satisfies well-foundedness and the finite anti-chain property. Both of these properties can be expressed using monadic second order variables. But, it is shown in [17] that Presburger Arithmetic with a single monadic variable becomes undecidable. Hence, this cannot directly be used to check if a Presburger-definable ordering is a *wqo*. However, we still have the following result:

▶ **Proposition 10.** *The property for a Presburger relation on $\mathbb{N}$ to be a well quasi ordering is decidable.*

3 The well structured problem for PCM

In the sequel, whenever we talk about PCM being WSTS, we will consider the usual ordering on $Q \times \mathbb{N}^d$ defined in subsection 2.2. We introduce a general technique to prove undecidability of checking whether a counter machine of some class is a WSTS. Let S_0 be the class of machines we are interested in. We will show reduction from reachability in Minsky machines.

▶ **Lemma 11.** *Suppose we have a procedure which takes a 2 counter Minsky machine with initial state $M = (Q, \rightarrow, q_0)$ and a control-state q_1 as input and generates a machine N of class S_0 which satisfies the following two requirements:*

- *All control-states in M are reachable implies N is a WSTS.* (1)
- *N is a WSTS implies q_1 is reachable in M from $(q_0; 0, 0)$.* (2)

Then, the well structured problem for S_0 is undecidable.

Proof. Suppose that the well structured problem for S_0 is decidable. We will use the above procedure to get an algorithm for Minsky machine reachability. Fix (M, q_1), where $M = (Q, \rightarrow_M, q_0)$. We want to check if q_1 is reachable from $(q_0; 0, 0)$.

Let $|Q| = n$. Consider all 2^{n-2} subsets $Q' \subseteq Q$ satisfying that $\{q_0, q_1\} \subseteq Q'$. For each such Q', let $\rightarrow_{Q'}$ denote the restriction of $\rightarrow_M$ to the set $Q' \times Q'$. Hence, we can associate a Minsky machine $M' = (Q', \rightarrow_{Q'}, q_0)$ to each such subset Q'. We call M' a *sub-machine* of M corresponding to Q'.

Now, for each sub-machine M', we consider the machine N' of class S_0, generated by the given procedure from (M', q_1). If there exists M' such that N' is a WSTS, then we have that q_1 is reachable in M' (by condition (2)), hence in M.

On the other hand, if q_1 was reachable in M, then let $Q_{reach} \subseteq Q$ be the set of all control-states of M which are reachable from $(q_0; 0, 0)$. Let its corresponding sub-machine be M'. Since all control-states of M' are reachable (by choice of Q_{reach}), therefore the corresponding N' will be a WSTS (by condition (1)).

Hence, q_1 is reachable in M from $(q_0; 0, 0)$ iff there exists a subset $Q' \subseteq Q$ satisfying that $\{q_0, q_1\} \subseteq Q'$ such that the corresponding sub-machine M' is a WSTS. Since there are only 2^{n-2} such subsets, we can check all of them to decide whether q_1 is reachable in M.

Hence, we have given an algorithm to check reachability in Minsky machine. Therefore, the well structured problem for S_0 is undecidable. ◀

We will use Lemma 11 to prove that the well structured problem for functional 1-dim PCMs is undecidable. To apply Lemma 11, we need to give an algorithm which takes a Minsky machine $M = (Q, \rightarrow_M, q_0)$ and a control-state q_1, and generates a functional 1-dim PCM N_1 satisfying conditions (1) and (2).

Construction of a functional 1-dim PCM N_1

Let (M, q_0) be given. The procedure to generate a 1-dim PCM N_1 is as follows:

Let $v_p(n)$ denote the largest power of p dividing n. For $M = (Q, \rightarrow_M, q_0)$, we define the 1-PCM $N_1 = (Q, \rightarrow_N, (q_0, 1))$ with the same set Q of control-states. We will represent the values of the two counters (m, n) by the one-counter values $2^m 3^n c$ for any c such that $v_2(c) = v_3(c) = 0$. Conversely, a configuration (q, n) of N_1 will correspond to $(q; v_2(n), v_3(n))$ of M. Note that, we are allowing multiplication by constants c in N_1 as long as $v_2(n)$ and $v_3(n)$ remain unchanged.

Increment/decrement of counters corresponds to multiplication/division by 2 and 3 which is Presburger expressible. Similarly, zero-test corresponds to checking divisibility by 2 and 3 which is again Presburger-expressible. So first, for each transition in $\rightarrow_M$, we add the corresponding transition to $\rightarrow_N$.

Now, to get the suitable properties of conditions (1) and (2), we will add two more types of transitions to $\rightarrow_N$. For each control-state q, we add a transition $(q, x_1' = 6x_1 + 1, q_0)$ to $\rightarrow_N$. We shall call it a *"reset-transition"* because $v_2(6x_1 + 1) = v_3(6x_1 + 1) = 0$, so this transition corresponds to a counter-reset in M from anywhere regardless of our present configuration. Note that such a transition would not change the reachability set in M. This "reset-transition" is crucial in forcing well-structuredness in N. Also, we add a transition $(q_0, (x_1 = 0 \wedge x_1' = 0), q_1)$ to $\rightarrow_N$ to ensure condition (2). Since the configuration $(q_0, 0)$ cannot be reached from the initial configuration $(q_0, 1)$ during any run of N_1, this will also not affect the reachability set of N_1. Note that, all of our transitions are functional, hence N_1 is a functional 1-dim PCM.

Now, we show that the construction of N_1 satisfies conditions (1) and (2).

▶ **Lemma 12.** *The functional 1-dim PCM N_1 satisfies condition (1).*

Proof. Suppose that all control-states of M are reachable from $(q_0; 0, 0)$. Then we claim that N_1 will be a WSTS. Suppose there is a transition $(q, n) \rightarrow_N (q', m)$ and (q, n') is a configuration with $(q, n') \geq (q, n)$. Hence we want to show existence of some path $(q, n') \xrightarrow{*}_N (q', m') \geq (q', m)$.

Case 1: The transition $(q, n) \rightarrow_N (q', m)$ is a "reset-transition". Hence $q' = q_0$ and $m = 6n+1$. In this case, note that since $n' \geq n$, the transition $(q, n') \rightarrow_N (q_0, 6n' + 1) \geq (q_0, m)$ satisfies the requirement.

Case 2: The transition $(q, n) \rightarrow_N (q', m)$ is not a "reset-transition". In this case, $m \leq 3n$ because the above transition corresponds, in M to an increment/decrement in c_1 or c_2 or a zero-test. In each case, we can check that $m \leq 3n$. Let there be a path $(q_0; 0, 0) \xrightarrow{*}_M (q'; n_1, n_2)$ in M for some n_1, n_2. Such a path exists because all control-states in M are reachable. Hence, we take the "reset-transition" $(q, n') \rightarrow_N (q_0, 6n' + 1)$ and follow the corresponding path $(q_0, 6n' + 1) \xrightarrow{*}_N (q', 2^{n_1} 3^{n_2} (6n' + 1)) \geq (q', 3n) \geq (q', m)$. Hence we have again shown monotony to prove that N_1 is a WSTS.

Hence we have shown that if all control-states of M are reachable, then N_1 is monotone. ◀

▶ **Lemma 13.** *The functional 1-dim PCM N_1 satisfies condition (2).*

Proof. Since there is a transition $(q_0, 0) \rightarrow_N (q_1, 0)$, we deduce that if N_1 is a WSTS, then $(q_0, 1) \xrightarrow{*}_N (q_1, n)$ for some n by monotony because $(q_0, 0) \leq (q_0, 1)$. Also note that since N_1 simulates M, hence reachability of q_1 in N_1 implies that q_1 is reachable from $(q_0; 0, 0)$ in M. ◀

Since we have provided a construction of functional 1-dim PCM N_1 satisfying conditions (1) and (2), from Lemma 11 we have that:

▶ **Theorem 14.** *The well structured problem for functional 1-dim PCMs is undecidable.*

Similarly, we can use Lemma 11 to show this result for 2 counter Minsky machines. This construction can be found in the extended version of this paper.

▶ **Theorem 15.** *The well structured problem for* 2*-dim Minsky machines is undecidable.*

Now, we make the observation that we can perform zero-tests using affine functions. The basic idea is that a transition $x' = -x$ is only satisfied by a counter whose value is 0. Increments/decrements can already be implemented in 2-AVASS since translations are affine functions. A zero test on the first counter can be done by having a transition labelled by $\left(\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$, and similarly for second counter. Since we can implement both increment/decrements and zero-tests with 2-AVASS, we can simulate 2-counter Minsky machines with 2-AVASS. Note that we can extend this result to d-AVASS simulating d-counter Minsky machines.

As a direct consequence of this and Theorem 15, we have that:

▶ **Corollary 16.** *The well structured problem for* 2*-AVASS is undecidable.*

However, if we consider strong monotony instead of monotony, the above undecidability results can be turned into a decidability result. Strong monotony can be expressed in Presburger arithmetic as follows:

$$\bigwedge_{\phi\in\Phi} (\forall x_1 ... \forall x_d \forall x'_1 ... \forall x'_d \forall y_1 ... \forall y_d ((\bigwedge_{i=1}^{d} x_i \leq y_i) \wedge \phi(x_1, ..., x_d, x'_1, ..., x'_d)$$
$$\implies (\exists y'_1 ... \exists y'_d (\bigwedge_{i=1}^{d} x'_i \leq y'_i) \wedge \phi(y_1, ..., y_d, y'_1, ..., y'_d))))$$

Since Presburger arithmetic is decidable, the strong well structured problem for d-PCM is decidable.

▶ Remark 17. The validity of the formula of strong monotony can also be decided for extended PCM defined in decidable extensions of Presburger Arithmetic.

4 Decidability results for 1-AVASS

Now, let us look at some reachability and coverability results for the various models of AVASS. First, we can simulate 2-counter Minsky machines with 2-AVASS. Since coverability and reachability are undecidable for 2-counter Minsky machines, we directly have the following result:

▶ **Corollary 18.** *Control-state reachability, hence coverability is undecidable for* 2*-AVASS.*

Similarly, we showed in Construction of functional 1-PCM N_1 that we can also simulate 2-counter Minsky machines with functional 1-PCM. Hence, we also have the following:

▶ **Corollary 19.** *Control-state reachability, hence coverability is undecidable for functional* 1*-PCM.*

Now, let us examine the case of 1-AVASS. For 1-AVASS, reachability and consequently coverability is decidable from work done in [10]. We show that checking whether it is a WSTS is also decidable. Moreover, we give a simpler proof of decidability of reachability and coverability.

Given $M = (Q, \rightarrow)$ a 1-AVASS and a final configuration (q_f, n_f) that we want to check reachability for, we present Algorithm 1 which computes $Pre^*(q_f, n_f)$ as a Presburger formula. A transition $(q, x' = ax + b, q')$ is *positive* if $a \geq 0$. Let a cycle/path in M be called *positive* if all transitions are positive. A cycle $(q_1, ..., q_k, q_1)$ is called a *simple cycle* if $q_1, ..., q_k$ are all pairwise distinct.

Let us denote by Pre_q the set $Pre^*(q_f, n_f) \cap (\{q\} \times \mathbb{N})$. For a transition $t = (q, x' = ax + b, q')$ and a given subset of $X \subseteq \mathbb{N}$, let $Pre^t(X)$ denote $\{n : an + b \in X\}$. For a simple cycle c rooted at q with an effective guard and transition, extend the above notation $Pre^{c^i}(X)$ for i repetitions of the cycle. Then, let $Pre^{c^*}(X) := \cup_{i \in \mathbb{N}} Pre^{c^i}(X)$. We will conveniently replace X by a formula which denotes a subset of $\mathbb{N}$.

Algorithm 1 Algorithm for computing $Pre^*(q_f, n_f)$ in 1-AVASS.

```
procedure COMPUTEPRE*
    for all $q \in Q$ do
        $\phi_q \equiv \bot$
    $\phi_{q_f} \equiv (n = n_f)$
    for all $q \in Q$ do
        for all simple cycles $c$ rooted at $q$ do
            $c$.transition = SIMPLIFYTRANSITION($c$)
            $c$.guard = COMPUTEGUARD($c$)
    notFinished = True
    while notFinished do
        notFinished = False
        for all $q \in Q$ do
            $\phi' = \phi_q$
            for all transitions $t = (q, x' = ax + b, q') \in\rightarrow$ do
                EXPLORETRANSITION($t$)
            for all simple cycles $c$ containing $q$ do
                EXPLORECYCLE($c$)
            if $\phi' \neq \phi_q$ then                    ▷ Check equality as Presburger formulae
                notFinished = True
```

The algorithm will keep a variable ϕ_q for each control-state $q \in Q$ which will store a Presburger formula (with one free variable n) denoting the currently discovered subset of Pre_q. Let this be denoted by $[\![\phi_q]\!]$, i.e. $[\![\phi_q]\!] := \{n : \phi_q(n)\}$. For uniformity, we can assume that ϕ_q is a disjunction of formulae of form $range \wedge mod$ where $range \equiv (r \leq n \leq s)$ (s possibly ∞) and $mod \equiv (n =_{d_q} d)$.

We initially simplify each simple cycle into a meta-transition which is the composition of all individual transitions in the cycle. We will also compute the guard of a cycle. Since each positive transition has an upward closed guard and each negative transition has a downward closed guard, the guard of a cycle will be of the form $r \leq n \leq s$ for some $r, s \in \mathbb{N}$ (s possibly ∞). Hence, we will only consider a cycle in terms of its guard and its meta-transition.

We use two main procedures in COMPUTEPRE*:

1. EXPLORETRANSITION: Given a transition $t = (q, x' = ax + b, q')$, it computes $Pre^t(\phi_{q'})$ and appends it to ϕ_q.
2. EXPLORECYCLE: Given a simple cycle c rooted at q, it computes $Pre^{c^*}(\phi_q)$ and appends it to ϕ_q.

▶ **Lemma 20.** *For any transition t, and any simple cycle c, given ϕ_q, $Pre^t(\phi_q)$ and $Pre^{c^*}(\phi_q)$ are both Presburger expressible and effectively computable.*

With this lemma the algorithm is well-defined. Now let us prove the termination and the correctness of the algorithm.

▶ **Proposition 21.** *Algorithm* COMPUTEPRE* *terminates.*

Proof. For each q, we will show that Pre_q can be obtained in finitely many iterations of the algorithm. Let $q \in Q$ be arbitrary.

Case 1: Pre_q is finite:
Each value will be discovered in finitely many iterations, hence Pre_q will be obtained in finitely many iterations.

Case 2: Pre_q is infinite:
Since we are talking about reaching (q_f, n_f), we note that the only transitions which can decrease arbitrarily large values are transitions of the form $x' = b$ or $x' = x - a, a > 0$. Hence, since Pre_q has arbitrarily large values, and each run has to reach n_f (i.e. has to be decreased), we can see that there must either be a transition $x' = b$, or a positive cycle with meta-transition $x' = x - a$, reachable from q through a positive path.

Case 2.i: There is a transition $x' = b$:
In this case, there exists N such that for all $n \geq N$, the same path suffices. In this case, once the aforementioned path is discovered, $\{n : n \geq N\}$ becomes a subset of $[\![\phi_q]\!] \subseteq Pre_q$, which leaves finitely many values in $Pre_q \setminus [\![\phi_q]\!]$, which can again be discovered by finitely many additional runs.

Case 2.ii: There are positive cycles with meta-transition $x' = x - a$:
The idea is that we will cover Pre_q when we compute Pre^{c^*} for such a cycle c. This is because for such a cycle, all that matters is the value of the counter modulo a. Since there are only finitely many distinct values modulo a, these will again be discovered in finitely many runs. Hence, each cycle will be discovered in finitely many runs. Therefore since there are finitely many simple cycles, the corresponding values of Pre_q will also be discovered in finitely many runs.

Hence, for all q, in finitely many runs we will get $Pre_q = [\![\phi_q]\!]$. At such a point, the algorithm has to stop, hence termination is guaranteed. ◀

▶ **Theorem 22** (Correctness). *Given a 1-AVASS $M = (Q, \rightarrow)$ and a configuration (q, n), the algorithm* COMPUTEPRE* *computes $Pre^*(q, n)$ as a Presburger formula.*

Proof. We will show that Algorithm 1 upon termination will always have $[\![\phi_q]\!] = Pre_q$.

That $[\![\phi_q]\!] \subseteq Pre_q$ should be clear. Suppose the algorithm terminates with $[\![\phi_q]\!] \subsetneq Pre_q$ for some $q \in Q$. For some value $n \in Pre_q \setminus [\![\phi_q]\!]$, consider a path which covers (q_2, n_2), say the path is $(q, n) \rightarrow (p_1, n_1) \rightarrow ... \rightarrow (p_m, n_m) \rightarrow (q_2, n')$. In such a path, consider the largest i, such that $n_i \notin [\![\phi_{p_i}]\!]$. Now, in the last iteration of the algorithm, since $n_{i+1} \in [\![\phi_{p_{i+1}}]\!]$ (by choice of i), hence, we will explore the edge to include $n_i \in [\![\phi_{p_i}]\!]$. Hence, the algorithm would not have terminated. Contradiction. Hence, when the algorithm terminates, $[\![\phi_q]\!] = Pre_q$. ◀

▶ **Example 23.** Let us consider machine M_1 in Figure 1. Suppose we want to compute $Pre^*(q_1, 19)$. We begin with $\phi_{q_1} \equiv (n = 19)$, $\phi_{q_2} \equiv \bot$. If we apply ExploreTransition to the transition $(q_2, (x' = x), q_1)$, we will get $\phi_{q_2} \equiv (n = 19)$. If we now apply ExploreCycle to the cycle $(q_2, x' = x - 3, q_2)$, we will get $\phi_{q_2} \equiv (n \geq 19 \wedge n =_3 1)$. Continuing like this, we end up with $\phi_{q_1} \equiv (n \in \{0, 3, 6, 19\} \vee (n \geq 13 \wedge n =_3 1) \vee (n \geq 32 \wedge n =_3 2) \vee (n \geq 45 \wedge n =_3 0))$ and $\phi_{q_2} \equiv (n \geq 0 \wedge n =_3 0) \vee (n \geq 19 \wedge n =_3 1) \vee (n \geq 32 \wedge n =_3 2)$. This is $Pre^*(q_1, 19)$.

▶ **Corollary 24** ([10])**.** *Reachability (hence coverability and control-state reachability) for* 1*-AVASS is decidable.*

▶ Remark 25. Algorithm 1 also works if we extend the model of 1-AVASS with Presburger guards at each transition. Hence, reachability, coverability and the well-structured problem are all decidable for this model as well.

It could be useful to determine whether an 1-AVASS is a WSTS (with strict monotony) because if it is the case, it will allow to decide other problems like the boundedness problem that is not immediately a consequence of the computability of $Pre^*(\uparrow(q, n))$. Since we can compute $Pre^*(q, n)$, we can also compute $Pre^*(\uparrow(q, n))$ by the same technique as in Corollary 24 This can be used to determine whether a given 1-AVASS is a WSTS as follows.

▶ **Theorem 26.** *The well structured problem is decidable for* 1*-AVASS.*

Proof. First we show that M is a WSTS, iff for all negative transitions $(q_1, (x' = ax + b), q_2)$, the set $\{q_1\} \times \mathbb{N}$ is a subset of $Pre^*(\uparrow(q_2, b))$. For any negative transition $(q_1, (x' = ax+b), q_2)$, we have $(q_1, 0) \rightarrow (q_2, b)$. If M is a WSTS, by monotony, for any $n \geq 0$, there exists a path $(q_1, n) \xrightarrow{*} (q_2, b') \geq (q_2, b)$ because $(q_1, n) \geq (q_1, 0)$. This implies that $\{q_1\} \times \mathbb{N}$ is a subset of $Pre^*(\uparrow(q_2, b))$.

In the other direction, let there be a transition $(q_1, n) \rightarrow (q_2, an + b)$ and $(q_1, n') \geq (q_1, n)$. If the transition is positive, i.e. $a \geq 0$, then we directly have the transition $(q_1, n') \rightarrow (q_2, an' + b) \geq (q_2, an + b)$. If the transition is negative, then we have that $(q_2, an + b) \leq (q_2, b)$. Since $(q_1, n') \in Pre^*(\uparrow(q_2, b))$ (by hypothesis, since it is a negative transition), hence we have that $(q_1, n') \xrightarrow{*} (q_2, b') \geq (q_2, b) \geq (q_2, an + b)$. Hence, M is monotone. Therefore, M is a WSTS iff for all negative transitions $(q_1, (x' = ax + b), q_2)$, the set $\{q_1\} \times \mathbb{N}$ is a subset of $Pre^*(\uparrow(q_2, b))$.

Now, since $Pre^*(\uparrow(q, n))$ is computable, we can check that for each negative transition $(q_1, (x' = ax + b), q_2)$, the set $\{q_1\} \times \mathbb{N}$ is a subset of $Pre^*(\uparrow(q_2, n))$ to determine whether M is a WSTS or not. ◀

▶ **Example 27.** Let us consider machine M_1 in Figure 1 and its negative transition $(q_1, x' = 19 - x, q_1)$. We observe that the set $Pre^*(\uparrow(q_1, 19)) = \{q_1, q_2\} \times \{n : n \geq 19\}$ does not contain $\{q_1\} \times \mathbb{N}$, hence machine M_1 is not a WSTS. However, in this example (Figure 1), if we replace the transition $(q_1, (x' = x - 13), q_2)$ by $(q_1, (x' = x + 1), q_2)$, we will get a new machine M_2 which is still not a 1-VASS, but it is a WSTS.

Let us focus our attention to positive AVASS now. We know that for positive 1-AVASS reachability is decidable from Corollary 24. We show that reachability is undecidable for positive 2-AVASS by reduction from Post's Correspondence Problem (PCP) [16]. Our result completes the view about decidability of reachability for VASS extensions in small dimensions. As a matter of fact, reachability is undecidable for VASS with two resets in dimension 3 (to adapt the proof in [7]), hence for positive 3-AVASS but it is decidable for VASS with two resets in dimension 2 [12]. If we replace resets by affine functions, reachability becomes undecidable in dimension two.

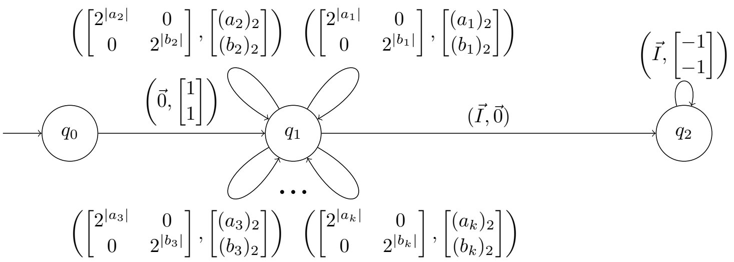

$\left(\begin{bmatrix}2^{|a_2|} & 0\\ 0 & 2^{|b_2|}\end{bmatrix}, \begin{bmatrix}(a_2)_2\\(b_2)_2\end{bmatrix}\right)$
$\left(\begin{bmatrix}2^{|a_1|} & 0\\ 0 & 2^{|b_1|}\end{bmatrix}, \begin{bmatrix}(a_1)_2\\(b_1)_2\end{bmatrix}\right)$
$\left(\vec{I}, \begin{bmatrix}-1\\-1\end{bmatrix}\right)$
$\left(\vec{0}, \begin{bmatrix}1\\1\end{bmatrix}\right)$
q_0
q_1
$(\vec{I},\vec{0})$
q_2
...
$\left(\begin{bmatrix}2^{|a_3|} & 0\\ 0 & 2^{|b_3|}\end{bmatrix}, \begin{bmatrix}(a_3)_2\\(b_3)_2\end{bmatrix}\right)$
$\left(\begin{bmatrix}2^{|a_k|} & 0\\ 0 & 2^{|b_k|}\end{bmatrix}, \begin{bmatrix}(a_k)_2\\(b_k)_2\end{bmatrix}\right)$

Figure 2 Construction for undecidability of reachability for positive 2-AVASS by reduction from PCP.

Reichert gives in [20] a reduction from the Post correspondence problem to reachability in a subclass of 2-AVASS and we may remark that his proof is still valid for *positive* 2-AVASS. Blondin, Haase and Mazowiecki made some similar observations [1] for subclasses of $3-\mathbb{Z}$-AVASS, with positive matrices. Our proof is essentially the same as [20].

▶ **Theorem 28.** *Reachability is undecidable for positive* 2*-AVASS.*

Proof. Suppose we are given an instance of PCP, i.e. we are given $a_1, ..., a_k, b_1, ..., b_k \in \{0,1\}^*$ for some $k \in \mathbb{N}$. We want to check if there exists some sequence of numbers $n_1, ..., n_\ell \in \{1, ..., k\}$ such that $a_{n_1}...a_{n_\ell} = b_{n_1}...b_{n_\ell}$ (concatenated as strings).

We will construct the positive 2-AVASS as demonstrated in Figure 2, where $|a_i|$ refers to the length of the string, and $(a_i)_2$ refers to the number encoded by the string a_i if read in binary (most significant digit to the left). The idea is that we use the two counters to store the value of $(a_{n_1}...a_{n_\ell})_2$ and $(b_{n_1}...b_{n_\ell})_2$ for any $n_1, ..., n_\ell$. But we first increment each counter to keep track of leading zeroes. Now, the configuration $(q_2; 0, 0)$ is reachable from $(q_0; 0, 0)$ in the positive 2-AVASS described in Figure 2 iff the given PCP has an affirmative answer. Hence, checking reachability in positive d-AVASS is undecidable for $d \geq 2$. ◀

Also, we note that positive-AVASS are well-structured with strong monotony. Hence coverability is decidable [13]. If we look at totally-positive AVASS, we can see that coverability is already decidable by the same argument. However, reachability is also decidable.

▶ **Theorem 29.** *Reachability is decidable in totally-positive AVASS for any dimension.*

Proof. Let $M = (Q, \rightarrow)$ be a totally-positive d-AVASS. Given $(q_0; n_1, ..., n_d)$, suppose we want to check reachability of $(q_f; m_1, ..., m_d)$. Let $N = \max\{m_1, ..., m_d\}$. Let $f_N : \mathbb{N} \rightarrow \{1, ..., N, \omega\}$ be the function which is identity on $\{1, ..., N\}$ and maps $\{N+1, ...\}$ to ω. Extend this function to the set $\mathbb{N}^d$ component-wise. Since M is totally-positive, we can restrict our search space from $Q \times \mathbb{N}^d$ to $Q \times \{0, ..., N, \omega\}^d$ by applying f_N to each configuration and using the following arithmetic rules: $0.\omega = 0$, and for all $k \geq 1$, $k.\omega = \omega$ and $\omega + k = \omega$.

We claim that if $(q_f; m_1, ..., m_d)$ is reachable, then it is reachable in this restricted search-space. This follows from the fact that given any element $(n_1, ..., n_d)$ of $\mathbb{N}^d$, and a totally positive transition $t = (A, b)$, we will have that $t(f_N(n_1, ..., n_d)) = f_N(t(n_1, ..., n_d))$ (t acts on $f_N(n_1, ..., n_d)$ to give an element in $\{0, ..., N, \omega\}^d$). This is because a totally positive transition cannot decrease a value other than by multiplying it by 0, hence any value greater than N will continue to be greater than N. Also note that, by choice of N, $f_N(m_1, ..., m_d) = (m_1, ..., m_d)$.

Once we have this, we can make an induction on the length of the path to see that if $(q_f; m_1, ..., m_d)$ is reachable, it is reachable in the restricted search-space $Q \times \{0, ..., N\}^d$.

Since $Q \times \{0, ..., N, \omega\}^d$ is finite, this shows decidability of reachability. ◀

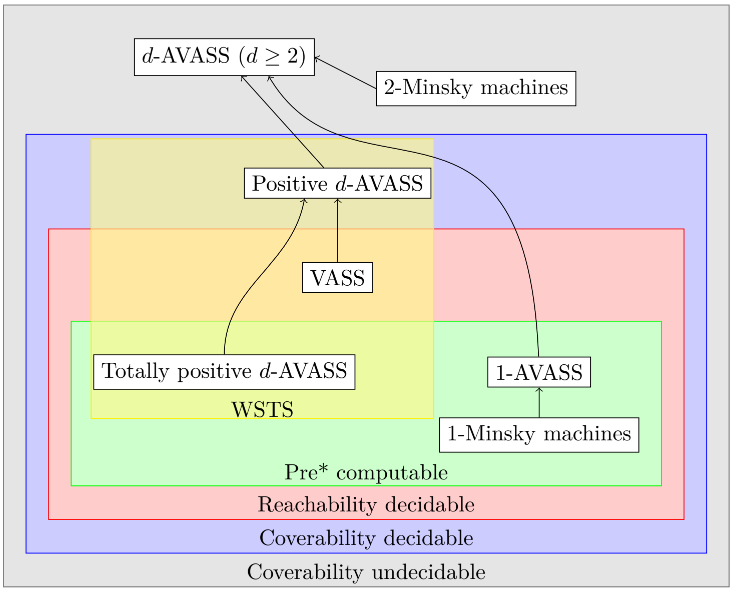


Figure 3 Showing reachability and coverability results for various AVASS models.

5 Conclusion and perspective

We introduced two variants of the well structured problem for PCM and we solve it for many classes of PCMs. Moreover, we answer the decidability questions for reachability and coverability for classes of PCMs and AVASSs (we summarise the results of Section 4 in Figure 3).

Many open problems can be attacked like the complexity of reachability for 1-AVASS (reachability is NP for 1-VASS and PSPACE for polynomial VASS), the size of Pre^* of a 1-AVASS (and its relation with the theory of flattable VASS [18]), and the decidability of the property for a Presburger relation on $\mathbb{N}^d$ to be a well-quasi ordering for $d \geq 2$.

We also open the way to study the decidability of the well structured problems (for various orderings) for many other models like pushdown counter machines, FIFO automata, Petri nets extensions. For instance, we wish to solve the *well structured problems* for FIFO automata. We know that *lossy* FIFO automata are well structured (for the subword ordering) but what is the class of *perfect* FIFO automata which is well structured (for the prefix ordering)?

References

1 Michael Blondin, Christoph Haase, and Filip Mazowiecki. Affine Extensions of Integer Vector Addition Systems with States. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.14.

2 Michael Blondin and Mikhail Raskin. The Complexity of Reachability in Affine Vector Addition Systems with States. *CoRR*, 2019. arXiv:1909.02579.

3 Bernard Boigelot and Pierre Wolper. Symbolic Verification with Periodic Sets. In *Computer Aided Verification, 6th International Conference, CAV '94, Stanford, California, USA, June 21-23, 1994, Proceedings*, pages 55–67, 1994. doi:10.1007/3-540-58179-0_43.

4 Rémi Bonnet, Alain Finkel, and M. Praveen. Extending the Rackoff technique to Affine nets. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 301–312. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.301.

5 Ahmed Bouajjani, Javier Esparza, Alain Finkel, Oded Maler, Peter Rossmanith, Bernard Willems, and Pierre Wolper. An Efficient Automata Approach to some Problems on Context-Free Grammars. *Information Processing Letters*, 74(5-6):221–227, June 2000. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/BEFMRWW-IPL2000.ps.

6 Stéphane Demri, Alain Finkel, Valentin Goranko, and Govert van Drimmelen. Model-checking CTL* over flat Presburger counter systems. *Journal of Applied Non-Classical Logics*, 20(4):313–344, 2010. doi:10.3166/jancl.20.313-344.

7 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset Nets between Decidability and Undecidability. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115, Aalborg, Denmark, July 1998. Springer. doi:10.1007/BFb0055044.

8 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1):63–92, 2001. ISS. doi:10.1016/S0304-3975(00)00102-X.

9 Alain Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.

10 Alain Finkel, Stefan Göller, and Christoph Haase. Reachability in Register Machines with Polynomial Updates. In Krishnendu Chatterjee and Jiří Sgall, editors, *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*, volume 8087 of *Lecture Notes in Computer Science*, pages 409–420, Klosterneuburg, Austria, August 2013. Springer. doi:10.1007/978-3-642-40313-2_37.

11 Alain Finkel and Jérôme Leroux. How to Compose Presburger-Accelerations: Applications to Broadcast Protocols. In *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science, 22nd Conference Kanpur, India, December 12-14, 2002, Proceedings*, pages 145–156, 2002. doi:10.1007/3-540-36206-1_14.

12 Alain Finkel, Jérôme Leroux, and Grégoire Sutre. Reachability for Two-Counter Machines with One Test and One Reset. In Sumit Ganguly and Paritosh Pandya, editors, *Proceedings of the 38th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'18)*, Leibniz International Proceedings in Informatics, pages 31:1–31:14, Ahmedabad, India, December 2018. Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2018.31.

13 Alain Finkel, Pierre McKenzie, and Claudine Picaronny. A Well-Structured Framework for Analysing Petri Net Extensions. *Information and Computation*, 195(1-2):1–29, November 2004. doi:10.1016/j.ic.2004.01.005.

14 Christoph Haase. A Survival Guide to Presburger Arithmetic. *ACM SIGLOG News*, 5(3):67–82, July 2018. doi:10.1145/3242953.3242964.

15 Christoph Haase and Simon Halfon. Integer Vector Addition Systems with States. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, Sept. 22–24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2014. doi:10.1007/978-3-319-11439-2_9.

16 Vesa Halava. Another proof of undecidability for the correspondence decision problem - Had I been Emil Post. *CoRR*, abs/1411.5197, 2014. arXiv:1411.5197.

17 Matthias Horbach, Marco Voigt, and Christoph Weidenbach. The Universal Fragment of Presburger Arithmetic with Unary Uninterpreted Predicates is Undecidable. *CoRR*, abs/1703.01212, 2017. arXiv:1703.01212.

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

19 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the Coverability Problem for Pushdown Vector Addition Systems in One Dimension. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 324–336, 2015. doi:10.1007/978-3-662-47666-6_26.

20 Julien Reichert. *Reachability games with counters : decidability and algorithms. (Décidabilité et complexité de jeux d'accessibilité sur des systèmes à compteurs)*. PhD thesis, École normale supérieure de Cachan, France, 2015. URL: https://tel.archives-ouvertes.fr/tel-01314414.

21 Rüdiger Valk. Self-Modifying Nets, a Natural Extension of Petri Nets. In Giorgio Ausiello and Corrado Böhm, editors, *Automata, Languages and Programming, Fifth Colloquium, Udine, Italy, July 17-21, 1978, Proceedings*, volume 62 of *Lecture Notes in Computer Science*, pages 464–476. Springer, 1978. doi:10.1007/3-540-08860-1_35.

A Categorical Account of Replicated Data Types

Fabio Gadducci
Dipartimento di Informatica, Università di Pisa, Italia
https://www.di.unipi.it/~gadducci
fabio.gadducci@unipi.it

Hernán Melgratti
Departamento de Computación, Universidad de Buenos Aires, Argentina
ICC-CONICET-UBA, Buenos Aires, Argentina
https://lafhis.dc.uba.ar/~melgratti
hmelgra@dc.uba.ar

Christian Roldán
Departamento de Computación, Universidad de Buenos Aires, Argentina
https://lafhis.dc.uba.ar/~croldan
croldan@dc.uba.ar

Matteo Sammartino
Department of Computer Science, University College London, UK
https://matteosammartino.com
m.sammartino@ucl.ac.uk

Abstract

Replicated Data Types (RDTs) have been introduced as a suitable abstraction for dealing with weakly consistent data stores, which may (temporarily) expose multiple, inconsistent views of their state. In the literature, RDTs are commonly specified in terms of two relations: visibility, which accounts for the different views that a store may have, and arbitration, which states the logical order imposed on the operations executed over the store. Different flavours, e.g., operational, axiomatic and functional, have recently been proposed for the specification of RDTs. In this work, we propose an algebraic characterisation of RDT specifications. We define categories of visibility relations and arbitrations, show the existence of relevant limits and colimits, and characterize RDT specifications as functors between such categories that preserve these additional structures.

2012 ACM Subject Classification Theory of computation → Program semantics; Software and its engineering → General programming languages

Keywords and phrases Replicated data type, Specification, Functorial characterisation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.42

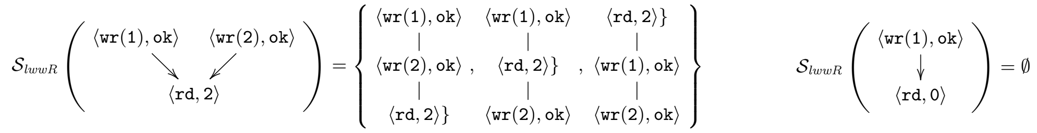
Funding Research partially supported by the MIUR PRIN 2017FTXR7S "IT-MaTTerS", by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement 778233, by the UBACyT projects 20020170100544BA and 20020170100086BA, by the PIP project 11220130100148CO, by the Leverhulme Prize PLP-2016-129, and by the EPSRC grant EP/S028641/1.

Acknowledgements We thank the reviewers for their careful reading and insightful comments.

1 Introduction

The CAP theorem establishes that a distributed data store can simultaneously provide two of the following three properties: consistency, availability, and tolerance to network partitions [8]. A weakly consistent data store prioritises availability and partition tolerance over consistency. As a consequence, a weakly consistent data store may (temporarily) expose multiple, inconsistent views of its state; hence, the behaviour of operations may depend on the particular view over which they are executed. Replicated data types (RDTs) have been proposed as suitable data type abstractions for weakly consistent data stores. The specification of such data types usually takes into account the particular views over which

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 42; pp. 42:1–42:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


$\mathcal{S}_{lwwR}$
⟨wr(1), ok⟩
⟨wr(2), ok⟩
⟨rd, 2⟩
=
⟨wr(1), ok⟩ ⟨wr(2), ok⟩ ⟨rd, 2⟩
⟨wr(1), ok⟩ ⟨rd, 2⟩ ⟨wr(2), ok⟩
⟨rd, 2⟩ ⟨wr(1), ok⟩ ⟨wr(2), ok⟩
$\mathcal{S}_{lwwR}$
⟨wr(1), ok⟩
⟨rd, 0⟩
= ∅

(a) Visibility relation with admissible arbitrations.

(b) Non admissible arbitrations.

Figure 1 A register specification.

operations are executed. A view is usually represented by a *visibility* relation, which is a binary, acyclic relation over the operations (a.k.a. *events*) executed by the system. The state of a store is described instead as a total order over the events, called *arbitration*, which describes the way in which conflicting concurrent operations are resolved. Different specification approaches for RDTs are presented in the literature, all of them building on the notions of visibility and arbitration [2, 3, 4, 5, 7, 6, 9, 11, 13, 14]. A purely functional approach for the specification of RDTs has been presented in [7, 6], where an RDT is associated with a function that maps each visibility relation into a set of arbitrations.

Consider an RDT `Register` that represents a memory cell, whose content can be updated and read. Following the approach in [7], the RDT `Register` is specified by a function that maps visibility relations into sets of arbitrations: we call here such function $\mathcal{S}_{lwwR}$. Figure 1a illustrates the definition of $\mathcal{S}_{lwwR}$ for the case in which the visibility relation involves two concurrent writes and a read. Events are depicted by pairs $\langle \mathtt{operation}, \mathtt{result}\rangle$ where $\mathtt{wr}(\mathtt{k})$ stands for an operation that writes the value $\mathtt{k}$ and $\mathtt{rd}$ stands for a read. The two writes are unrelated (i.e., they are not visible to each other), while the read operation sees both writes. The returned value of the read operation is $\mathtt{2}$, which coincides with one of the visible written values. According to Figure 1a, $\mathcal{S}_{lwwR}$ maps such visibility graph into a set containing those arbitrations (i.e., total orders over the three events in the visibility relation) in which $\mathtt{wr}(\mathtt{1})$ precedes $\mathtt{wr}(\mathtt{2})$. Arbitrations may not reflect the causal ordering of events; in fact, the last two arbitrations in the right-hand-side of the equation in Figure 1a place the read before the operation that writes the read value $\mathtt{2}$. We remark that arbitrations do not necessarily account for real-time orderings of events: they are instead possible ways in which events can be *logically* ordered to explain a given visibility. For instance, the excluded arbitrations in the image of $\mathcal{S}_{lwwR}$ are the total orders in which $\mathtt{wr}(\mathtt{2})$ precedes $\mathtt{wr}(\mathtt{1})$, i.e., the specification bans the behaviour in which a read operation returns a value that is different from the last written one. An extreme situation is the case in which the specification maps a visibility relation into an empty set of arbitrations, which means that events cannot be logically ordered to explain such visibility. For instance, the equation in Figure 1b assigns an empty set of arbitrations to a visibility relation in which the read operation returns a value that is different from the unique visible written value (i.e., it returns 0 instead of 1). In this way, the specification bans the behaviour in which a read operation returns a value that does not match a previous written value. As originally shown [7], this style of specification can be considered (and it is actually more general than) the model for the operational description of RDTs proposed in [4]. We refer the reader to [6] for a formal comparison of the two different approaches.

This work develops the approach suggested in [7] for the categorical characterisation of RDT specifications. We consider the category $\mathbf{PIDag}(\mathcal{L})$ of labelled, directed acyclic graphs and injective *pr-morphisms*, i.e., label-preserving morphisms that reflect directed edges, and the category $\mathbf{SPath}(\mathcal{L})$ of sets of labelled, total orders and *ps-morphisms*, i.e., morphisms between sets of paths. A ps-morphism $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ from a set of paths $\mathcal{X}_1$ to a set of paths $\mathcal{X}_2$ states that any total order in $\mathcal{X}_2$ can be obtained by extending some total order in $\mathcal{X}_1$. In this work we show that a large class of specifications, dubbed *coherent*, can be characterised functorially. Roughly, a coherent specification accounts for those RDTs such that the arbitrations associated with a visibility relation can be obtained by extending arbitrations associated with "smaller" visibilities: as illustrated in [6], they correspond to what are called *return value consistent* RDTs in [4]. We establish a bijection between functors and specifications, showing that a coherent specification induces a functor from $\mathbf{PIDag}(\mathcal{L})$ into $\mathbf{SPath}(\mathcal{L})$ that preserves colimits and binary pullbacks and vice versa.

The paper has the following structure. Section 2 offers some preliminaries on categories of relations, which are used for proposing some basic results on categories of graphs and paths in Section 3. Section 4 recalls the set-theoretical presentation of RDTs introduced in [6]. Section 5 introduces our semantical model, the category of set of paths, describing some of its basic properties with respect to limits and colimits. In Section 6 we present some categorical operators for RDTs, which are used in Section 7 to present our main characterisation results. The paper is closed with some final remarks, a comparison of the proposed constructions with those presented in [7], and some hints towards future work.

2 Preliminaries on Relations

Relations. Given a finite set $\mathtt{E}$, a (binary) *relation* ρ over $\mathtt{E}$ is a subset $\rho \subseteq \mathtt{E} \times \mathtt{E}$ of the cartesian product of $\mathtt{E}$ with itself. We use the pair $\langle \mathtt{E}, \rho \rangle$ to denote a relation ρ over $\mathtt{E}$, in order to always have the set of events explicit, and simply $\emptyset$ to denote the empty relation.

A subset $\mathtt{E}' \subseteq \mathtt{E}$ is *downward closed* with respect to ρ if $\forall \mathtt{e} \in \mathtt{E}, \mathtt{e}' \in \mathtt{E}'.\mathtt{e}\ \rho\ \mathtt{e}'$ implies $\mathtt{e} \in \mathtt{E}'$ and, when ρ is clear, we write $\lfloor \mathtt{e} \rfloor$ for the smallest downward closed set including $\mathtt{e} \in \mathtt{E}$.

▶ **Definition 1** ((Binary Relation) Morphisms). *A (binary relation) morphism* $\mathtt{f} : \langle \mathtt{E}, \rho \rangle \to \langle \mathtt{T}, \gamma \rangle$ *is a function* $\mathtt{f} : \mathtt{E} \to \mathtt{T}$ *such that*

$$\forall e, e' \in \mathtt{E}.\ e\ \rho\ e' \textit{ implies } \mathtt{f}(e)\ \gamma\ \mathtt{f}(e')$$

A morphism $\mathtt{f} : \langle \mathtt{E}, \rho \rangle \to \langle \mathtt{T}, \gamma \rangle$ *is* past-reflecting *(shortly, pr-morphism) if*

$$\forall e \in \mathtt{E}, \mathtt{t} \in \mathtt{T}.\ \mathtt{t}\ \gamma\ \mathtt{f}(e) \textit{ implies } \exists e' \in \mathtt{E}.\ e'\ \rho\ e\ \wedge\ \mathtt{t} = \mathtt{f}(e')$$

Note that both classes of morphisms are closed under composition: we denote as **Bin** the category of relations and their morphisms and **PBin** the sub-category of pr-morphisms.

▶ **Lemma 2** (Characterising pr-morphisms). *Let* $\mathtt{f} : \langle \mathtt{E}, \rho \rangle \to \langle \mathtt{T}, \gamma \rangle$ *be a morphism. If*

1. $\mathtt{f}(e)\ \gamma\ \mathtt{f}(e')$ *implies* $e\ \rho\ e'$*, and*
2. $\bigcup_{e \in \mathtt{E}} \mathtt{f}(e)$ *is downward closed,*

then it is a pr-morphism. If $\mathtt{f}$ *is injective, then the converse holds.*

Proof. For $\Rightarrow$), let us take $\mathtt{e} \in \mathtt{E}$ and $\mathtt{t} \in \mathtt{T}$. If $\mathtt{t}\ \gamma\ \mathtt{f}(\mathtt{e})$, then there exists $\mathtt{e}' \in \mathtt{E}$ such that $\mathtt{t} = \mathtt{f}(\mathtt{e}')$ because of (2). By (1), $\mathtt{f}(\mathtt{e}')\ \gamma\ \mathtt{f}(\mathtt{e})$ implies $\mathtt{e}'\ \rho\ \mathtt{e}$.

For $\Leftarrow$), by the definition of pr-morphism $\mathtt{f}(\mathtt{e})\ \gamma\ \mathtt{f}(\mathtt{e}')$ implies $\exists \bar{\mathtt{e}} \in \mathtt{E}.\ \bar{\mathtt{e}}\ \rho\ \mathtt{e}'\ \wedge\ \mathtt{f}(\mathtt{e}) = \mathtt{f}(\bar{\mathtt{e}})$. Since $\mathtt{f}$ is injective, $\bar{\mathtt{e}} = \mathtt{e}$ and hence $\mathtt{e}\ \rho\ \mathtt{e}'$. So, let $\mathcal{T} = \bigcup_{\mathtt{e} \in \mathtt{E}} \mathtt{f}(\mathtt{e})$. We want to show that

$$\forall \mathtt{t} \in \mathtt{T}, \mathtt{t}' \in \mathcal{T}.\ \mathtt{t}\gamma\ \mathtt{t}' \text{ implies } \mathtt{t} \in \mathcal{T}$$

The proof follows by contradiction. Assume that $\exists \mathtt{t} \in \mathtt{T}, \mathtt{t}' \in \mathcal{T}.\ \mathtt{t}\ \gamma\ \mathtt{t}'\ \wedge\ \mathtt{t} \notin \mathcal{T}$. By definition of $\mathcal{T}, \exists \mathtt{e} \in \mathtt{E}$ such that $\mathtt{f}(\mathtt{e}) = \mathtt{t}'$. Since $\mathtt{f}$ is a pr-morphism, then

$$\mathtt{t}\ \gamma\ \mathtt{f}(\mathtt{e}) \text{ implies } \exists \mathtt{e}' \in \mathtt{E}.\ \mathtt{e}'\ \rho\ \mathtt{e}\ \wedge\ \mathtt{t} = \mathtt{f}(\mathtt{e}')$$

Therefore $\mathtt{t} = \mathtt{f}(\mathtt{e}') \in \mathcal{T}$, which contradicts the assumption $\mathtt{t} \notin \mathcal{T}$. ◀

Clearly, **Bin** has both finite limits and finite colimits, which are computed point-wise as in **Set**. The structure is largely lifted to **PBin**.

▶ **Proposition 3** (Properties of **PBin**). *The inclusion functor* **PBin** $\to$ **Bin** *reflects finite colimits and binary pullbacks.*

In other words, since **Bin** has finite limits and finite colimits, finite colimits and binary pullbacks in **PBin** always exist and are computed as in **Bin**. There is e.g. no terminal object, since morphisms in **Bin** into the singleton are clearly not past-reflecting.

Monos in **Bin** are just morphisms whose underlying function is injective, and similarly in **PBin**, so that the inclusion functor preserves (and reflects) them.

▶ **Lemma 4** (Monos under pushouts). *Pushouts in* **Bin** *(and thus in* **PBin***) preserve monos.*

We now introduce labelled relations. Consider the forgetful functors $\mathtt{U_r} : \mathbf{Bin} \to \mathbf{Set}$ and $\mathtt{U_p} : \mathbf{PBin} \to \mathbf{Set}$, the latter factoring through the inclusion functor **PBin** $\to$ **Bin**. Given a set $\mathcal{L}$ of labels, we consider the comma categories $\mathbf{Bin}(\mathcal{L}) = \mathtt{U_r} \downarrow \mathcal{L}$ and $\mathbf{PBin}(\mathcal{L}) = \mathtt{U_p} \downarrow \mathcal{L}$: finite colimits and binary pullbacks always exist and are essentially computed as in **Bin**.

Explicitly, an object in $\mathtt{U_r} \downarrow \mathcal{L}$ is a triple $(\mathtt{E}, \rho, \lambda)$ for a labeling function $\lambda : \mathtt{E} \to \mathcal{L}$. A label-preserving morphism $(\mathtt{E}, \rho, \lambda) \to (\mathtt{E}', \rho', \lambda')$ is a morphism $\mathtt{f} : (\mathtt{E}, \rho) \to (\mathtt{E}', \rho')$ such that $\forall \mathtt{s} \in \mathtt{E}.\ \lambda(\mathtt{s}) = \lambda'(\mathtt{f}(\mathtt{s}))$. Moreover, finite colimits and binary pullbacks exist and are computed as in **Bin**. Similar properties hold for the objects and the morphisms of $\mathtt{U_p} \downarrow \mathcal{L}$.

3 Categories of Graphs and Paths

We now move to introduce specific sub-categories that are going to be used for both the syntax and the semantics of specifications.

▶ **Definition 5** (**PDag**). **PDag** *is the full sub-category of* **PBin** *whose objects are directed acyclic graphs.*

In other terms, objects are relations whose transitive closures are *strict* partial orders.

▶ Remark 6. The full sub-category of **Bin** whose objects are directed acyclic graphs is not suited for our purposes, since e.g. it does not admit pushouts, not even along monos. The one with pr-morphisms is much more so, still remaining computationally simple.

▶ **Proposition 7** (Properties of **PDag**). *The inclusion functor* **PDag** $\to$ **PBin** *reflects finite colimits and binary pullbacks.*

We now move to consider *paths*, i.e., relations that are total orders.

▶ **Definition 8** (**Path**). **Path** *is the full sub-category of* **Bin** *whose objects are paths.*

Note that defining **Path** as only containing pr-morphisms would be too restrictive, since there exists a pr-morphism between two paths if and only if one path is a prefix of the other.

▶ **Proposition 9** (Properties of **Path**). *The inclusion functor* **Path** $\to$ **Bin** *reflects finite colimits.*

As for relations, we consider suitable comma categories in order to capture labelled paths and graphs. In particular, we use the forgetful functors $\mathtt{U_{rp}} : \mathbf{Path} \to \mathbf{Set}$ and $\mathtt{U_{pd}} : \mathbf{PDag} \to \mathbf{Set}$: for a set of labels $\mathcal{L}$ we denote $\mathbf{PDag}(\mathcal{L}) = \mathtt{U_{rp}} \downarrow \mathcal{L}$ and $\mathbf{Path}(\mathcal{L}) = \mathtt{U_{pd}} \downarrow \mathcal{L}$. Once more, finite colimits and binary pullbacks always exist and are essentially computed as in **Bin**.

4 Replicated Data Type Specification

We briefly recall the set-theoretical model of replicated data types (RDT) introduced in [6]. Our main result is its categorical characterisation, which is given in the following sections.

First, some notation. We denote a graph as the triple $\langle \mathcal{E}, \prec, \lambda \rangle$ and a path as the triple $\langle \mathcal{E}, \leq, \lambda \rangle$, in order to distinguish them. Moreover, given a graph $\mathtt{G} = \langle \mathcal{E}, \prec, \lambda \rangle$ and a subset $\mathcal{E}' \subseteq \mathcal{E}$, we denote by $\mathtt{G}|_{\mathcal{E}'}$ the obvious restriction (and the same for a path $\mathtt{P}$).

We now define a product operation on a set of paths $\mathcal{X} = \{\langle \mathcal{E}_i, \leq_i, \lambda_i \rangle\}_i$. First, we say that the paths of a set $\mathcal{X}$ are *compatible* if $\forall \mathtt{e}, i, j.\ \mathtt{e} \in \mathcal{E}_i \cap \mathcal{E}_j$ implies $\lambda_i(\mathtt{e}) = \lambda_j(\mathtt{e})$.

▶ **Definition 10** (Product). *Let $\mathcal{X}$ be a set of compatible paths. The product of $\mathcal{X}$ is*

$$\bigotimes \mathcal{X} = \{\mathtt{P} \mid \mathtt{P} \ \textit{ is a path over } \bigcup_i \mathcal{E}_i \textit{ and } \mathtt{P}|_{\mathcal{E}_i} \in \mathcal{X} \ \}.$$

Intuitively, the product of paths is analogous to the synchronous product of transition systems, in which common elements are identified and the remaining ones can be freely interleaved, as long as the original orders are respected. A set of sets of paths $\mathcal{X}_1, \mathcal{X}_2, \ldots$ is compatible if $\bigcup_i \mathcal{X}_i$ is so. In such case we can define the product $\bigotimes_i \mathcal{X}_i$ as $\bigotimes \bigcup_i \mathcal{X}_i$.

Now, let us further denote with $\mathbb{G}(\mathcal{L})$ and $\mathbb{P}(\mathcal{L})$ the sets of (finite) graphs and (finite) paths, respectively, labelled over $\mathcal{L}$ and with ϵ the empty graph. Also, when the set of labels $\mathcal{L}$ is chosen, we let $\mathbb{G}(\mathcal{E}, \lambda)$ and $\mathbb{P}(\mathcal{E}, \lambda)$ the sets of graphs and paths, respectively, whose elements are those in $\mathcal{E}$ and are labelled by $\lambda : \mathcal{E} \to \mathcal{L}$.

▶ **Definition 11** (Specifications). *A* specification *$\mathcal{S}$ is a function $\mathcal{S} : \mathbb{G}(\mathcal{L}) \to 2^{\mathbb{P}(\mathcal{L})}$ such that $\mathcal{S}(\epsilon) = \{\epsilon\}$ and $\forall \mathtt{G}.\ \mathcal{S}(\mathtt{G}) \in 2^{\mathbb{P}(\mathcal{E}_\mathtt{G}, \lambda_\mathtt{G})}$.*

In other words, a specification $\mathcal{S}$ maps a graph (interpreted in terms of the visibility relation of a RDT) to a set of paths (that is, the admissible arbitrations of the RDT). Indeed, note that $\mathtt{P} \in \mathcal{S}(\mathtt{G})$ is a path over $\mathcal{E}_\mathtt{G}$, hence a total order of the events in $\mathtt{G}$.

As shown in [6], Definition 11 offers an alternative characterisation of RDTs [4] for a suitable choice of the set of labels. In particular, an RDT boils down to a specification labelled over pairs $\langle \mathit{operation}, \mathit{value} \rangle$ that is *saturated* and *past-coherent*. The former property is a technical one: roughly, if $\mathtt{G}'$ is an extension of $\mathtt{G}$ with a fresh event $\mathtt{e}$, then the admissible arbitrations that a saturated specification $\mathcal{S}$ assigns to $\mathtt{G}'$ (i.e., the set of paths $\mathcal{S}(\mathtt{G}')$) are included in the admissible arbitrations of $\mathtt{G}$ saturated with respect to $\mathtt{e}$, i.e., all the paths that extends a path in $\mathcal{S}(\mathtt{G})$ with $\mathtt{e}$ inserted at an arbitrary position. Coherence instead is fundamental and expresses that admissible arbitrations of a visibility graph can be obtained by composing the admissible arbitrations of smaller visibilities.

▶ **Definition 12** ((Past-)Coherent Specification). *Let $\mathcal{S}$ be a specification. We say that $\mathcal{S}$ is past-coherent (briefly, coherent) if*

$$\forall \mathtt{G} \neq \epsilon.\ \mathcal{S}(\mathtt{G}) = \bigotimes_{e \in \mathcal{E}_\mathtt{G}} \mathcal{S}(\mathtt{G}|_{\lfloor \mathtt{e} \rfloor}).$$

Explicitly, in a coherent specification $\mathcal{S}$ the arbitrations of a configuration $\mathtt{G}$ (i.e., the set of paths $\mathcal{S}(\mathtt{G})$) are the composition of the arbitrations associated with its sub-graphs $\mathtt{G}|_{\lfloor \mathtt{e} \rfloor}$.

Next example illustrates a coherent specification for the `Register` RDT.

▶ **Example 13** (Register)**.** Fix the set of labels $\mathcal{L} = \{\langle \mathtt{wr}(\mathtt{k}), \mathtt{ok}\rangle, \langle \mathtt{rd}, \mathtt{k}\rangle \mid \mathtt{k} \in \mathbb{N}\} \cup \{\langle \mathtt{rd}, \bot\rangle\}$. Then, the specification of the RDT `Register` is given by the function $\mathcal{S}_{lwwR}$ defined as

$$\mathtt{P} \in \mathcal{S}_{lwwR}(\mathtt{G}) \;\text{ iff }\; \forall \mathtt{e} \in \mathcal{E}_{\mathtt{G}}. \begin{cases} \lambda(\mathtt{e}) = \langle \mathtt{rd}, \bot\rangle \text{ implies } \forall \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}, \mathtt{k}.\ \lambda(\mathtt{e}') \neq \langle \mathtt{wr}(\mathtt{k}), \mathtt{ok}\rangle \\ \forall \mathtt{k}.\ \lambda(\mathtt{e}) = \langle \mathtt{rd}, \mathtt{k}\rangle \text{ implies } \exists \mathtt{e}' \prec_{\mathtt{G}} \mathtt{e}.\ \lambda(\mathtt{e}') = \langle \mathtt{wr}(\mathtt{k}), \mathtt{ok}\rangle \text{ and} \\ \qquad \forall \mathtt{e}'' \prec_{\mathtt{G}} \mathtt{e}, \mathtt{k}' \neq \mathtt{k}.\ \mathtt{e}' <_{\mathtt{P}} \mathtt{e}'' \text{ implies } \lambda(\mathtt{e}'') \neq \langle \mathtt{wr}(\mathtt{k}'), \mathtt{ok}\rangle \end{cases}$$

Intuitively, a visibility graph $\mathtt{G}$ is mapped to a non-empty set of arbitrations (i.e., $\mathcal{S}_{lwwR}(\mathtt{G}) \neq \emptyset$) only when each event $\mathtt{e}$ in $\mathtt{G}$ associated with a read operation has a return value $\mathtt{k}$ that matches the value written by the greatest event $\mathtt{e}'$ (according to $<_{\mathtt{P}}$). The result of a read is undefined (i.e., $\bot$) when it does not see any write (first condition).

5 The model category

In order to provide a categorical characterisation of coherent specifications, we must first define precisely the model category. So far, we know that its objects have to be sets of compatible paths. We fix a set of labels $\mathcal{L}$, and we first look at a free construction for paths, and then we turn our attention to morphisms.

5.1 Saturation

▶ **Definition 14** (Path saturation)**.** *Let* $\mathtt{P}$ *be a path and* $\mathtt{f} : (\mathcal{E}_{\mathtt{P}}, \lambda_{\mathtt{P}}) \to (\mathcal{E}, \lambda)$ *a function preserving labels. The saturation of* $\mathtt{P}$ *along* $\mathtt{f}$ *is defined as*

$$\mathtt{sat}(\mathtt{P}, \mathtt{f}) = \{\mathtt{Q} \mid \mathtt{Q}\ \in \mathbb{P}(\mathcal{E}, \lambda) \textit{ and } \mathtt{f} \textit{ induces a morphism } \mathtt{f} : \mathtt{P} \to \mathtt{Q}\}$$

Saturation is generalised to sets of paths $\mathcal{X} \subseteq \mathbb{P}(\mathcal{E}, \lambda)$ *as* $\bigcup_{\mathtt{P} \in \mathcal{X}} \mathtt{sat}(\mathtt{P}, \mathtt{f})$.

Note that, should $\mathtt{f}$ not be injective, it could be that $\mathtt{sat}(\mathtt{P}, \mathtt{f}) = \emptyset$.

▶ **Example 15.** Consider the injective, label-preserving function $\mathtt{f}$ from $\{\langle \mathtt{wr}(1), \mathtt{ok}\rangle, \langle \mathtt{wr}(2), \mathtt{ok}\rangle\}$ to $\{\langle \mathtt{wr}(1), \mathtt{ok}\rangle, \langle \mathtt{wr}(2), \mathtt{ok}\rangle, \langle \mathtt{rd}, 2\rangle\}$. Then, we have

$$\mathtt{sat}\left(\left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \end{array} \right\}, \mathtt{f}\right) = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle\} \end{array}, \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle\} \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \end{array}, \begin{array}{c} \langle \mathtt{rd}, 2\rangle\} \\ | \\ \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \end{array} \right\}$$

Intuitively, saturation adds $\langle \mathtt{rd}, 2\rangle$ – and in general events not in the image of $\mathtt{f}$ – to the original path in all possible ways, preserving the order of original events.

▶ **Definition 16** (Path retraction)**.** *Let* $\mathtt{Q}$ *be a path and* $\mathtt{f} : \mathcal{E} \to \mathcal{E}_{\mathtt{Q}}$ *a function. The retraction of* $\mathtt{Q}$ *along* $\mathtt{f}$ *is defined as*

$$\mathtt{ret}(\mathtt{Q}, \mathtt{f}) = \{\mathtt{P} \mid \mathtt{P}\ \in \mathbb{P}(\mathcal{E}, \lambda) \textit{ and } \mathtt{f} \textit{ induces a morphism } \mathtt{f} : \mathtt{P} \to \mathtt{Q}\}$$

The notion of retraction is extended to sets of paths $\mathcal{X} \subseteq \mathbb{P}(\mathcal{E}, \lambda)$ *as* $\bigcup_{\mathtt{Q} \in \mathcal{X}} \mathtt{ret}(\mathtt{Q}, \mathtt{f})$.

Note that λ is fully characterised as the restriction of $\lambda_{\mathtt{Q}}$ along the mapping. Should $\mathtt{f}$ be injective, $\mathtt{ret}(\mathtt{Q}, \mathtt{f})$ would be a singleton, and if $\mathtt{f}$ is an inclusion, then $\mathtt{ret}(\mathtt{Q}, \mathtt{f}) = \mathtt{Q}|_{\mathcal{E}}$.

We may now start considering the relationship between the two notions.

▶ **Lemma 17.** *Let* $\mathcal{X}_1 \subseteq \mathbb{P}(\mathcal{E}_1, \lambda_1)$ *be a set of paths and* $\mathtt{f} : (\mathcal{E}_1, \lambda_1) \to (\mathcal{E}_2, \lambda_2)$ *a function preserving labels. Then* $\mathcal{X}_1 \subseteq \mathtt{ret}(\mathtt{sat}(\mathcal{X}_1, \mathtt{f}), \mathtt{f})$*. If* $\mathtt{f}$ *is injective, then the equality holds.*

▶ **Lemma 18.** *Let* $\mathcal{X}_2 \subseteq \mathbb{P}(\mathcal{E}_2, \lambda_2)$ *be a set of paths and* $\mathtt{f} : \mathcal{E}_1 \to \mathcal{E}_2$ *a function. Then* $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathtt{ret}(\mathcal{X}_2, \mathtt{f}), \mathtt{f})$*.*

We say that an injective function $\mathtt{f}$ is *saturated* with respect to $\mathcal{X}_2$ if the equality holds.

▶ **Example 19.** Consider the set of paths $\mathcal{X}_1$ and $\mathcal{X}_2$ and the pr-morphism $\mathtt{f}$ below

$$\mathcal{X}_1 = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \end{array} \right\} \qquad \mathcal{X}_2 = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array} \right\} \qquad \mathtt{f} : \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \end{array} \to \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array}$$

the underlying function $\mathtt{f}$ (defined in Example 15) is *not* saturated with respect to $\mathcal{X}_2$ because

$$\left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array} \right\} \neq \mathtt{sat}(\mathtt{ret}(\left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array} \right\}, \mathtt{f}), \mathtt{f}) = \mathtt{sat}(\left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \end{array} \right\}, \mathtt{f})$$

In fact, the ps-morphism $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ only adds the new event $\langle \mathtt{rd}, 2\rangle$ on top of the path in $\mathcal{X}_1$, thus making it a *topological* ps-morphism (see Section 7.3 later on).

5.2 From saturation to categories

We can exploit saturation to get a simple definition of our model category.

▶ **Definition 20** (ps-morphism)**.** *Let* $\mathcal{X}_1 \subseteq \mathbb{P}(\mathcal{E}_1, \lambda_1)$ *and* $\mathcal{X}_2 \subseteq \mathbb{P}(\mathcal{E}_2, \lambda_2)$ *be sets of paths. A path-set morphim (shortly, ps-morphism)* $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ *is a function* $\mathtt{f} : (\mathcal{E}_1, \lambda_1) \to (\mathcal{E}_2, \lambda_2)$ *preserving labels such that* $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathcal{X}_1, \mathtt{f})$*.*

Intuitively, there is a ps-morphism from the set of paths $\mathcal{X}_1$ to the set of paths $\mathcal{X}_2$ if any path in $\mathcal{X}_2$ can be obtained by adding events to some path in $\mathcal{X}_1$. This notion captures the idea that arbitrations of larger visibilities are obtained as extensions of smaller visibilities.

▶ **Example 21.** Consider the following three sets and the function $\mathtt{f}$ from Example 15

$$\mathcal{X}_1 = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \end{array} \right\} \quad \mathcal{X}_2 = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array} , \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \end{array} \right\} \quad \mathcal{X}_3 = \left\{ \begin{array}{c} \langle \mathtt{wr}(1), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \end{array} , \begin{array}{c} \langle \mathtt{wr}(2), \mathtt{ok}\rangle \\ | \\ \langle \mathtt{rd}, 2\rangle \\ | \\ \langle \mathtt{wr}(1), \mathtt{ok}\rangle \end{array} \right\}$$

Now, $\mathtt{f}$ induces a ps-morphism $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ because $\mathcal{X}_2 \subseteq \mathtt{sat}(\mathcal{X}_1, \mathtt{f})$ (the latter is shown in Example 15). On the contrary, there is no ps-morphism from $\mathcal{X}_1$ to $\mathcal{X}_3$: the rightmost path of $\mathcal{X}_3$ cannot be obtained by extending a path of $\mathcal{X}_1$ with an event labelled by $\langle \mathtt{rd}, 2\rangle$.

▶ **Definition 22** (Sets of Paths Category)**.** *We define* $\mathbf{SPath}(\mathcal{L})$ *as the category whose objects are sets of paths labelled over* $\mathcal{L}$ *and arrows are ps-morphisms.*

▶ **Proposition 23** (Properties of $\mathbf{SPath}$)**.** *The category* $\mathbf{SPath}(\mathcal{L})$ *has finite colimits along monos and binary pullbacks.*

Proof.

(Strict) initial object. The (unique) initial object is $\langle\emptyset,\{\epsilon\},\emptyset\rangle$, with $\epsilon\in\mathbb{P}(\emptyset,\emptyset)$ the empty path. Let $\mathcal{X}\subseteq\mathbb{P}(\mathcal{E},\lambda)$ and $!\colon\emptyset\to\mathcal{E}$ the unique function. We have a function $!\colon(\emptyset,\emptyset)\to(\mathcal{E},\lambda)$ such that $\mathcal{X}\subseteq\mathtt{sat}(\{\epsilon\},!)=\mathbb{P}(\mathcal{E},\lambda)$.

Binary Pushouts. Let $\mathcal{X},\mathcal{X}_1$, and $\mathcal{X}_2$ be sets of paths and $\mathtt{f}_i:\mathcal{X}\to\mathcal{X}_i$ ps-morphisms. Consider the underlying functions $\mathtt{f}_i:\mathcal{E}\to\mathcal{E}_i$ and their pushout $\mathtt{f}'_i:\mathcal{E}_i\to\mathcal{E}_1+_{\mathcal{E}}\mathcal{E}_2$ in the category of sets: it induces a pushout $\mathtt{f}'_i:\mathcal{X}_i\to\mathtt{sat}(\mathcal{X}_1,\mathtt{f}'_1)\cap\mathtt{sat}(\mathcal{X}_2,\mathtt{f}'_2)$ in $\mathbf{SPath}(\mathcal{L})$.

Binary Pullbacks. Let $\mathcal{X},\mathcal{X}_1$, and $\mathcal{X}_2$ be sets of paths and $\mathtt{f}_i:\mathcal{X}_i\to\mathcal{X}$ ps-morphisms. Consider the underlying functions $\mathtt{f}_i:\mathcal{E}_i\to\mathcal{E}$ and their pullback $\mathtt{f}'_i:\mathcal{E}_1\times_{\mathcal{E}}\mathcal{E}_2\to\mathcal{E}_i$ in the category of sets: it induces a pullback $\mathtt{f}'_i:\mathtt{ret}(\mathcal{X}_1,\mathtt{f}'_1)\cup\mathtt{ret}(\mathcal{X}_2,\mathtt{f}'_2)\to\mathcal{X}_i$ in $\mathbf{SPath}(\mathcal{L})$. ◀

The above characterisation of pushouts is enabled by the fact that we considered injective functions. To help intuition, we now instantiate that characterisation to suitable inclusions.

▶ **Lemma 24.** *Let $\mathtt{f}_i:\mathcal{X}\to\mathcal{X}_i$ be ps-morphisms such that the underlying functions $\mathtt{f}_i:\mathcal{E}\to\mathcal{E}_i$ are inclusions and $\mathcal{E}=\mathcal{E}_1\cap\mathcal{E}_2$. Then their pushout is given by $\mathtt{f}'_i:\mathcal{X}_i\to\mathcal{X}_1\otimes\mathcal{X}_2$.*

Proof. By definition $\mathcal{X}_1\otimes\mathcal{X}_2=\{\mathtt{P}\mid\mathtt{P}\ \text{is a path over}\ \bigcup_i\mathcal{E}_i\ \text{and}\ \mathtt{P}|_{\mathcal{E}_i}\in\mathcal{X}_i\}$. Note also that $\mathtt{sat}(\mathcal{X}_i,\mathtt{f}'_\mathtt{i})=\bigcup_{\mathtt{Q}\in\mathcal{X}_i}\{\mathtt{P}\mid\mathtt{P}\ \in\mathbb{P}(\bigcup_i\mathcal{E}_i,\bigcup_i\lambda_i)$ and $\mathtt{f}'_\mathtt{i}$ induces a path morphism $\mathtt{f}'_\mathtt{i}:\mathtt{P}\to\mathtt{Q}\}$. Since $\mathtt{f}'_\mathtt{i}$ is an inclusion, the latter condition equals to $\mathtt{P}|_{\mathcal{E}_i}=\mathtt{Q}$, thus the property holds. ◀

▶ **Example 25.** Consider the following ps-morphisms

$$\left\{\begin{array}{c}\langle\mathtt{wr}(1),\mathtt{ok}\rangle\\ |\\ \langle\mathtt{wr}(2),\mathtt{ok}\rangle\\ |\\ \langle\mathtt{rd},2\rangle\end{array}\right\}\leftarrow\left\{\begin{array}{c}\langle\mathtt{wr}(1),\mathtt{ok}\rangle\\ |\\ \langle\mathtt{wr}(2),\mathtt{ok}\rangle\end{array}\right\}\rightarrow\left\{\begin{array}{ccc}\langle\mathtt{wr}(1),\mathtt{ok}\rangle & & \langle\mathtt{wr}(2),\mathtt{ok}\rangle\\ | & & |\\ \langle\mathtt{wr}(2),\mathtt{ok}\rangle & , & \langle\mathtt{wr}(1),\mathtt{ok}\rangle\\ | & & |\\ \langle\mathtt{rd},1\rangle & & \langle\mathtt{rd},1\rangle\end{array}\right\}$$

then, the pushout is given by the following ps-morphisms

$$\left\{\begin{array}{c}\langle\mathtt{wr}(1),\mathtt{ok}\rangle\\ |\\ \langle\mathtt{wr}(2),\mathtt{ok}\rangle\\ |\\ \langle\mathtt{rd},2\rangle\end{array}\right\}\rightarrow\left\{\begin{array}{ccc}\langle\mathtt{wr}(1),\mathtt{ok}\rangle & & \langle\mathtt{wr}(1),\mathtt{ok}\rangle\\ | & & |\\ \langle\mathtt{wr}(2),\mathtt{ok}\rangle & & \langle\mathtt{wr}(2),\mathtt{ok}\rangle\\ | & , & |\\ \langle\mathtt{rd},1\rangle & & \langle\mathtt{rd},2\rangle\\ | & & |\\ \langle\mathtt{rd},2\rangle & & \langle\mathtt{rd},1\rangle\end{array}\right\}\leftarrow\left\{\begin{array}{ccc}\langle\mathtt{wr}(1),\mathtt{ok}\rangle & & \langle\mathtt{wr}(2),\mathtt{ok}\rangle\\ | & & |\\ \langle\mathtt{wr}(2),\mathtt{ok}\rangle & , & \langle\mathtt{wr}(1),\mathtt{ok}\rangle\\ | & & |\\ \langle\mathtt{rd},1\rangle & & \langle\mathtt{rd},1\rangle\end{array}\right\}$$

An analogous property holds for pullbacks. Let $\mathtt{f}_i:\mathcal{X}_i\to\mathcal{X}$ be ps-morphisms such that the underlying functions are inclusions: the pullback is given as $\mathtt{f}'_i:\bigcup_i\mathcal{X}_i|_{\mathcal{E}_1\cap\mathcal{E}_2}\to\mathcal{X}_i$. In particular, the square below is both a pullback and a pushout.

$$\begin{tikzcd}
\bigcup_i\mathcal{X}_i|_{\mathcal{E}_1\cap\mathcal{E}_2} \arrow[r, hook] \arrow[d, hook] & \mathcal{X}_1 \arrow[d, hook] \\
\mathcal{X}_2 \arrow[r, hook] & \mathcal{X}_1\otimes\mathcal{X}_2
\end{tikzcd}$$

6 Structure and Operators for Visibility

We now study the category of visibility relations. We first introduce an operation that will be handy for our categorical characterisation. We say that a graph $\mathtt{G}$ is *rooted* if there exists a (necessarily unique) event $\mathtt{e}\in\mathcal{E}_\mathtt{G}$ such that $\mathtt{G}=\mathtt{G}|_{\lfloor\mathtt{e}\rfloor}$.

▶ **Definition 26** (Extension). *Let* $\mathtt{G} = \langle \mathcal{E}, \prec, \lambda \rangle$ *and* $\mathcal{E}' \subseteq \mathcal{E}$. *We define the* extension *of* $\mathtt{G}$ *over* $\mathcal{E}'$ *with* ℓ *as the graph* $\mathtt{G}^{\ell}_{\mathcal{E}'} = \langle \mathcal{E}_\top, \prec \cup (\mathcal{E}' \times \{\top\}), \lambda[\top \mapsto \ell] \rangle$.

Here, $\mathcal{E}_\top$ denotes the extension of the set $\mathcal{E}$ with a new event $\top$, labelled into ℓ. Intuitively, $\mathtt{G}^{\ell}_{\mathcal{E}'}$ is obtained by adding to the visibility relation $\mathtt{G}$ a new event "seeing" some events in $\mathcal{E}'$. We call the inclusion $\mathtt{G} \to \mathtt{G}^{\ell}_{\mathcal{E}'}$ an *extension morphism.* Should $\mathtt{G}^{\ell}_{\mathcal{E}'}$ be rooted, we call it a *root* extension of $\mathtt{G}$, and the associated inclusion a root extension morphism.

▶ **Proposition 27.** *Rooted graphs form a family of separators of* $\mathbf{PDag}(\mathcal{L})$.

Proof. We need to show that for any pair of pr-morphisms $\mathtt{f}_1, \mathtt{f}_2 : \mathtt{G}_1 \to \mathtt{G}_2$ such that $\mathtt{f}_1 \neq \mathtt{f}_2$ there is a rooted graph $\mathtt{G}$ and a morphism $\mathtt{f} : \mathtt{G} \to \mathtt{G}_1$ such that $\mathtt{f};\mathtt{f}_1 \neq \mathtt{f};\mathtt{f}_2$. Given $\mathtt{e} \in \mathcal{E}_{\mathtt{G}_1}$ such that $\mathtt{f}_1(\mathtt{e}) \neq \mathtt{f}_2(\mathtt{e})$, it suffices to consider the pr-morphism $\mathtt{f} : \mathtt{G}_1|_{\lfloor \mathtt{e} \rfloor} \to \mathtt{G}_1$. ◀

We now further curb the arrows in $\mathbf{PDag}(\mathcal{L})$ to *monic* ones. Intuitively, we are only interested in what happens if we add further events to visibility relations. We thus consider the sub-category $\mathbf{PIDag}(\mathcal{L})$ of direct acyclic graphs and monic pr-morphisms. Note that the chosen morphism $\mathtt{f}$ in the proof of Proposition 27 is mono, since morphisms in $\mathbf{PDag}(\mathcal{L})$ are monic if and only if the underlying function is injective. We can then show that rooted graphs are also a family of generators for the sub-category $\mathbf{PIDag}(\mathcal{L})$.

We first need a technical lemma.

▶ **Lemma 28** (Monos under pushouts, 2). *Pushouts in* $\mathbf{PDag}(\mathcal{L})$ *preserve monos.*

We can then state an important characterisation of $\mathbf{PIDag}(\mathcal{L})$.

▶ **Proposition 29.** $\mathbf{PIDag}(\mathcal{L})$ *is the smallest sub-category of* $\mathbf{PDag}(\mathcal{L})$ *containing all root extension morphisms and closed under finite colimits.*

Proof. First, note that, since pushouts in $\mathbf{PDag}(\mathcal{L})$ preserve monos, the smallest sub-category of $\mathbf{PDag}(\mathcal{L})$ containing all root extensions and closed under finite colimits is surely a sub-category also of $\mathbf{PIDag}(\mathcal{L})$. So, given a monic pr-morphism $\mathtt{f} : \mathtt{G}_1 \to \mathtt{G}_2$, we need to prove that it can be generated from root extension morphisms via colimits. We proceed by induction on the cardinality of $\mathcal{E}_{\mathtt{G}_2}$.

If the cardinality is 0, then $\mathtt{f}$ must be the identity of the empty graph. Otherwise, consider $\mathtt{G}_2$ and assume that it is rooted with root $\mathtt{e}$. Now, if $\mathtt{e} \in \mathsf{img}(\mathtt{f})$, since the image of a pr-morphism is downward closed, it turns out that $\mathtt{f}$ is the identity of $\mathtt{G}_2$. If it is not in the image, then $\mathtt{f}$ can be decomposed as $\mathtt{G}_1 \to (\mathtt{G}_2 \setminus \mathtt{e}) \to \mathtt{G}_2$: the left-most is given by induction, while the right-most is a root extension morphism. Without loss of generality, let us assume that $\mathtt{G}_2$ has two distinct roots, namely $\mathtt{e}_1$ and $\mathtt{e}_2$, and that the image of $\mathtt{f}$ is contained in $\mathtt{G}_2|_{\lfloor \mathtt{e}_1 \rfloor}$. Now, $\mathtt{f}$ can be decomposed as $\mathtt{G}_1 \to \mathtt{G}_2|_{\lfloor \mathtt{e}_1 \rfloor} \to \mathtt{G}_2$: the left-most is given by induction, while the right-most is obtained via the pushout of the span $\mathtt{G}_2|_{\lfloor \mathtt{e}_1 \rfloor} \cap \mathtt{G}_2|_{\lfloor \mathtt{e}_2 \rfloor} \to \mathtt{G}_2|_{\lfloor \mathtt{e}_i \rfloor}$. ◀

7 A categorical correspondence

It is now the time for moving towards our categorical characterisation of specifications. In this section we will show that coherent specifications induce functors preserving the relevant categorical structure (soundness) and, conversely, that a certain class of functors (basically, those preserving finite colimits and binary pullbacks) induce coherent specifications (completeness). Finally, we will prove that these functions between functors and specifications are mutually inverse, establishing a one-to-one correspondence (up-to isomorphism).

We first provide a simple technical result for coherent specifications.

▶ **Lemma 30.** *Let $\mathcal{S}$ be a coherent specification and $\mathcal{E} \subseteq \mathcal{E}_{\mathtt{G}}$. If $\mathcal{E}$ is downward closed, then $\mathcal{S}(\mathtt{G})|_{\mathcal{E}} \subseteq \mathcal{S}(\mathtt{G}|_{\mathcal{E}})$.*

Proof. Let $\mathcal{E}$ be downward closed, and note that this amounts to requiring $\mathcal{E} = \bigcup_{e \in \mathcal{E}} \lfloor \mathtt{e} \rfloor$, hence for all $\mathtt{e} \in \mathcal{E}$ we have that $(\mathtt{G}|_{\mathcal{E}})|_{\lfloor \mathtt{e} \rfloor} = \mathtt{G}|_{\lfloor \mathtt{e} \rfloor}$. By the latter and by coherence we have $\mathcal{S}(\mathtt{G})|_{\mathcal{E}} = (\bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathcal{S}(\mathtt{G}|_{\lfloor \mathtt{e} \rfloor}))\Big|_{\mathcal{E}}$ and $\mathcal{S}(\mathtt{G}|_{\mathcal{E}}) = \bigotimes_{\mathtt{e} \in \mathcal{E}} \mathcal{S}(\mathtt{G}|_{\lfloor \mathtt{e} \rfloor})$. Note that $(\bigotimes_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathcal{S}(\mathtt{G}|_{\lfloor \mathtt{e} \rfloor}))\Big|_{\mathcal{E}} \subseteq \bigotimes_{\mathtt{e} \in \mathcal{E}} \mathcal{S}(\mathtt{G}|_{\lfloor \mathtt{e} \rfloor})$ because a path in the former can always be restricted to a suitable path with fewer events on the latter (the converse in general does not hold). ◀

7.1 Soundness

The notion of specification introduced in Definition 11 is oblivious to the existence of morphisms between graphs. In the following we impose a minimal consistency requirement, i.e., that a specification maps isomorphic graphs to isomorphic sets of paths, along the same isomorphism on events. That is, if there exists an isomorphism in **PDag** from $\mathtt{G}_1$ to $\mathtt{G}_2$ with underlying bijection $\mathtt{f} : \mathcal{E}_{\mathtt{G}_1} \to \mathcal{E}_{\mathtt{G}_2}$, then for all specifications $\mathcal{S}$ there is an isomorphism in $\mathbf{SPath}(\mathcal{L})$ from $\mathcal{S}(\mathtt{G}_1)$ to $\mathcal{S}(\mathtt{G}_2)$ with the same underlying function.

▶ **Proposition 31** (functors induced by specifications)**.** *A coherent specification $\mathcal{S}$ induces a functor $\mathbb{M}(\mathcal{S}) : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$.*

Proof. For $\mathtt{G}$ we define $\mathbb{M}(\mathcal{S})(\mathtt{G})$ as $\mathcal{S}(\mathtt{G})$ and for $\mathtt{f} : \mathtt{G} \to \mathtt{G}'$ we define $\mathbb{M}(\mathcal{S})(\mathtt{f})$ as the ps-morphism with underlying injective function $\mathtt{f} \colon (\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}}) \hookrightarrow (\mathcal{E}_{\mathtt{G}'}, \lambda_{\mathtt{G}'})$. The proof boils down to showing that $\mathtt{f}$ really is a ps-morphism from $\mathcal{S}(\mathtt{G})$ into $\mathcal{S}(\mathtt{G}')$, i.e., $\mathcal{S}(\mathtt{G}') \subseteq \mathtt{sat}(\mathcal{S}(\mathtt{G}), \mathtt{f})$ and, since we are considering specifications preserving isomorphisms, we can restrict our attention to the case where $\mathtt{f}$ is an inclusion.

Since $\mathtt{f}$ is a pr-morphism, $\bigcup_{\mathtt{e} \in \mathcal{E}_{\mathtt{G}}} \mathtt{f}(\mathtt{e})$ is downward-closed in $\mathtt{G}'$ and thus by Lemma 30 we have $\mathcal{S}(\mathtt{G}')|_{\mathcal{E}_{\mathtt{G}}} \subseteq \mathcal{S}(\mathtt{G}'|_{\mathcal{E}_{\mathtt{G}}}) = \mathcal{S}(\mathtt{G})$, the latter equality given by coherence. Now, consider a path $\mathtt{P} \in \mathcal{S}(\mathtt{G}')$. Since $\mathtt{P}|_{\mathcal{E}_{\mathtt{G}}} \in \mathcal{S}(\mathtt{G})$, we have $\mathtt{P} \in \mathtt{sat}(\mathcal{S}(\mathtt{G}), \mathtt{f})$, because saturation adds missing events – namely those in $\mathcal{E}_{\mathtt{G}'} \setminus \mathcal{E}_{\mathtt{G}}$ – to $\mathtt{P}|_{\mathcal{E}_{\mathtt{G}}}$ in all possible ways. Therefore we can conclude $\mathcal{S}(\mathtt{G}') \subseteq \mathtt{sat}(\mathcal{S}(\mathtt{G}), \mathtt{f})$. ◀

It is a well-known fact that the category of sets and injective functions lacks pushouts. The same also holds for $\mathbf{PIDag}(\mathcal{L})$. However, recall now that pushouts in $\mathbf{PDag}(\mathcal{L})$ preserve monos (Lemma 28). Thus in the following we say that a functor $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ weakly preserves finite pushouts (and in fact, finite colimits) if any commuting square in $\mathbf{PIDag}(\mathcal{L})$ that is a pushout (via the inclusion functor) in $\mathbf{PDag}(\mathcal{L})$ is mapped by $\mathbb{F}$ to a pushout in $\mathbf{SPath}(\mathcal{L})$.

▶ **Theorem 32.** *Let $\mathcal{S}$ be a coherent specification. The induced functor $\mathbb{M}(\mathcal{S}) : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ weakly preserves finite colimits and preserves binary pullbacks.*

Proof. The initial object is easy, since it holds by construction. As for pushouts and pullbacks: since $\mathcal{S}$ is coherent, it boils down to Lemma 24. ◀

7.2 Completeness

It is now time for moving to the completeness results of our work, showing (a few alternatives on) how to obtain a specification from a functor.

▶ **Theorem 33.** *Let $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ be a functor such that $\mathbb{F}(\mathtt{G}) \subseteq \mathbb{P}(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}})$. If $\mathbb{F}$ weakly preserves finite colimits and preserves binary pullbacks, it induces a coherent specification $\mathbb{S}(\mathbb{F})$.*

Proof. Let $\mathbb{S}(\mathbb{F})(\mathtt{G}) = \mathbb{F}(\mathtt{G})$. We shall show that $\mathbb{F}(\mathtt{G})$ is coherent. Consider the following pushout in $\mathbf{PDag}(\mathcal{L})$

$$\begin{tikzcd}
\mathtt{G}|_{\lfloor \mathtt{e}_1 \rfloor \cap \lfloor \mathtt{e}_2 \rfloor} \arrow[r, hook] \arrow[d, hook] & \mathtt{G}|_{\lfloor \mathtt{e}_2 \rfloor} \arrow[d, hook] \\
\mathtt{G}|_{\lfloor \mathtt{e}_1 \rfloor} \arrow[r, hook] & \mathtt{G}|_{\lfloor \mathtt{e}_1 \rfloor \cup \lfloor \mathtt{e}_2 \rfloor}
\end{tikzcd} \tag{7.1}$$

Since $\mathbb{F}$ preserves pullbacks, thus monos, and weakly preserves pushouts, this diagram is mapped by $\mathbb{F}$ to the following pushout in $\mathbf{SPath}(\mathcal{L})$

$$\begin{tikzcd}
\mathbb{F}(\mathtt{G}|_{\lfloor \mathtt{e}_1 \rfloor \cap \lfloor \mathtt{e}_2 \rfloor}) \arrow[r, hook] \arrow[d, hook] & \mathbb{F}(\mathtt{G}|_{\lfloor \mathtt{e}_2 \rfloor}) \arrow[d, hook] \\
\mathbb{F}(\mathtt{G}|_{\lfloor \mathtt{e}_1 \rfloor}) \arrow[r, hook] & \mathbb{F}(\mathtt{G}|_{\lfloor \mathtt{e}_1 \rfloor \cup \lfloor \mathtt{e}_2 \rfloor})
\end{tikzcd} \tag{7.2}$$

where all underlying functions between events are inclusions. By Lemma 24 we have that

$$\mathbb{F}(\mathtt{G}|_{\lfloor \mathtt{e}_1 \rfloor \cup \lfloor \mathtt{e}_2 \rfloor}) \simeq \mathbb{F}(\mathtt{G}|_{\lfloor \mathtt{e}_1 \rfloor}) \otimes \mathbb{F}(\mathtt{G}|_{\lfloor \mathtt{e}_2 \rfloor})$$

Since clearly $\mathtt{G} = \mathtt{G}|_{\bigcup_{\mathtt{e} \in \mathcal{E}_\mathtt{G}} \lfloor \mathtt{e} \rfloor}$, by associativity of pushouts we obtain coherence

$$\mathbb{F}(\mathtt{G}) \simeq \bigotimes_{\mathtt{e} \in \mathcal{E}_\mathtt{G}} \mathbb{F}(\mathtt{G}|_{\lfloor \mathtt{e} \rfloor})$$

Isomorphism preservation follows from $\mathbb{F}$ being a functor. ◀

Combined with Theorem 32, the result above intuitively tells us that the coherence of a specification roughly corresponds to the weak preservation of colimits. However, the set-theoretical requirement $\mathbb{F}(\mathtt{G}) \subseteq \mathbb{P}(\mathcal{E}_\mathtt{G}, \lambda_\mathtt{G})$ is still unsatisfactory, yet apparently unavoidable, because a generic $\mathbb{F}$ could associate *any* set of paths to a graph. We can sharpen the result by requiring functors to preserve specific properties for suitable arrows of $\mathbf{PIDag}(\mathcal{L})$. The candidates are root extension morphisms, given the properties shown in Section 6. In order to define the functors, we also need to consider a suitable subset of the arrows of $\mathbf{SPath}(\mathcal{L})$.

▶ **Definition 34** (Saturated specifications). *Let $\mathcal{S}$ be a specification. It is* saturated *if for all graphs $\mathtt{G}$ and extensions $\mathtt{G}^\ell_\mathcal{E}$ the inclusion $\mathtt{f} : \mathcal{E}_\mathtt{G} \to \mathcal{E}_{\mathtt{G}^\ell}$ is saturated with respect to $\mathcal{S}(\mathtt{G}^\ell_\mathcal{E})$ (see Lemma 18), that is*

$$\forall \mathtt{G}, \mathcal{E}, \ell.\ \mathcal{S}(\mathtt{G}^\ell_\mathcal{E}) = \mathtt{sat}(\mathtt{ret}(\mathcal{S}(\mathtt{G}^\ell_\mathcal{E}), \mathtt{f}), \mathtt{f})$$

A *saturation ps-morphism* (along ℓ) is a saturated ps-morphism $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ with underlying function $(\mathcal{E}, \lambda) \to (\mathcal{E}_\top, \lambda[\top \mapsto \ell])$. We can now prove an instance of Theorem 33 concerning saturated specifications.

▶ **Proposition 35.** *Let $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ be a functor mapping root extension morphisms into saturation ps-morphisms (along the same labels). If $\mathbb{F}$ weakly preserves finite colimits, it induces a saturated, coherent specification $\mathbb{S}(\mathbb{F})$.*

Proof. We first show that $\mathbb{F}$ preserves monos, which renders the assumption of Theorem 33 about preservation of pullbacks redundant. We will essentially follow the proof of Proposition 29. Given $\mathtt{f} \colon \mathtt{G}_1 \to \mathtt{G}_2$ in $\mathbf{PIDag}(\mathcal{L})$, we proceed by induction on the cardinality of $\mathcal{E}_{\mathtt{G}_2}$. If $\mathcal{E}_{\mathtt{G}_2}$ is $\emptyset$, i.e., it is the initial object, then $\mathtt{f}$ is the identity on $\emptyset$, and the claim follows by functors preserving identities. Suppose now that $\mathtt{G}_2$ is rooted with root $\mathtt{e}$. If $\mathtt{e} \in \mathsf{img}(\mathtt{f})$, then $\mathtt{f}$ again is the identity. Otherwise, $\mathtt{f}$ can be decomposed as $\mathtt{G}_1 \to (\mathtt{G}_2 \setminus \mathtt{e}) \to \mathtt{G}_2$: the left-most one satisfies the induction hypothesis, and the right-most one is a root extension morphism, which by hypothesis is mapped to a (monic) saturation ps-morphism. Therefore, by functoriality of $\mathbb{F}$, the claim holds for the composition of these morphisms. If $\mathtt{G}_2$ is not rooted, then $\mathtt{f}$ can be similarly decomposed as $\mathtt{G}_1 \to \mathtt{G}_2|_{\lfloor \mathtt{e}_1 \rfloor} \to \mathtt{G}_2$. By induction the claim holds for the left-most morphism. The right-most one is obtained via a pushout of the form (7.1), which is mapped by $\mathbb{F}$ to a pushout of the form (7.2), because $\mathbb{F}$ (weakly) preserves finite colimits. By induction hypothesis, the span of this pushout consists of monic ps-morphisms, therefore we use Lemma 24 to conclude that the pushout morphisms are monic as well, hence the right-most morphism satisfies our claim. Again, the claim for the whole of $\mathtt{f}$ follows from functoriality of $\mathbb{F}$. A similar inductive argument can be used to show that $\mathbb{F}(\mathtt{G})$ is a set of paths over $(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}})$ (up to a label-preserving isomorphism of events). Therefore we can now re-use the proof of Theorem 33 and obtain that $\mathbb{S}(\mathbb{F})$ is a coherent specification.

It remains to be shown that $\mathbb{S}(\mathbb{F})$ is saturated, that is $\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}) = \mathtt{sat}(\mathtt{ret}(\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}), \mathtt{f}), \mathtt{f})$. If $\mathtt{G}^{\ell}_{\mathcal{E}}$ is rooted, this follows from $\mathbb{F}$ mapping root extensions to saturation ps-morphisms. Otherwise, by coherence, $\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}})$ can be decomposed into the product $\bigotimes_{\mathtt{e} \in (\mathcal{E}_{\mathtt{G}})_{\top}} \mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}|_{\lfloor \mathtt{e} \rfloor})$. For each component of the product we have a root extension $\mathtt{G}^{\ell}_{\mathcal{E}}|_{\lfloor \mathtt{e} \rfloor} \setminus \mathtt{e} \to \mathtt{G}^{\ell}_{\mathcal{E}}|_{\lfloor \mathtt{e} \rfloor}$, which is mapped by $\mathbb{F}$ to a saturation ps-morphism, therefore we have $\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}|_{\lfloor \mathtt{e} \rfloor}) = \mathtt{sat}(\mathtt{ret}(\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}}|_{\lfloor \mathtt{e} \rfloor}), \mathtt{f}_{\mathtt{e}}), \mathtt{f}_{\mathtt{e}})$, where $\mathtt{f}_{\mathtt{e}}$ is the underlying function between events of the root extension. Saturation of $\mathbb{F}(\mathtt{G}^{\ell}_{\mathcal{E}})$ follows by computing the product of these sets of paths. ◀

7.3 More Completeness

The need of finding a suitable image for root extension morphisms allows for alternative choices. To this end, we introduce a different subset of the arrows of $\mathbf{SPath}(\mathcal{L})$.

▶ **Definition 36** (Path extension/prefixing). *Let* $\mathtt{P}$ *be a path and* $\mathtt{f} : (\mathcal{E}_{\mathtt{P}}, \lambda_{\mathtt{P}}) \to (\mathcal{E}, \lambda)$ *a function preserving labels. The extension of* $\mathtt{P}$ *along* $\mathtt{f}$ *is defined as*

$$\mathtt{ext}(\mathtt{P}, \mathtt{f}) = \{\mathtt{Q} \mid \mathtt{Q} \in \mathbb{P}(\mathcal{E}, \lambda) \textit{ and } \mathtt{f} \textit{ induces a pr-morphism } \mathtt{f} : \mathtt{P} \to \mathtt{Q}\}$$

Similarly, let $\mathtt{Q}$ *be a path and* $\mathtt{f} : \mathcal{E} \to \mathcal{E}_{\mathtt{Q}}$ *a function preserving labels. The prefixing of* $\mathtt{Q}$ *along* $\mathtt{f}$ *is defined as*

$$\mathtt{pre}(\mathtt{Q}, \mathtt{f}) = \{\mathtt{P} \mid \mathtt{P} \in \mathbb{P}(\mathcal{E}, \lambda) \textit{ and } \mathtt{f} \textit{ induces a pr-morphism } \mathtt{f} : \mathtt{P} \to \mathtt{Q}\}$$

Both definitions immediately extend to sets of paths. Should $\mathtt{f}$ be injective, $\mathtt{pre}(\mathtt{Q}, \mathtt{f})$ would be a singleton, and if $\mathtt{f}$ is an inclusion, then $\mathtt{pre}(\mathtt{Q}, \mathtt{f}) = \mathtt{Q}|_{\mathcal{E}}$, for the latter a prefix of $\mathtt{Q}$. Also, note that similarly $\mathtt{P}$ has to be a prefix for all the paths in $\mathtt{ext}(\mathtt{P}, \mathtt{f})$.

▶ **Example 37.** A topological specification $\mathcal{S}_{topR}$ for a `Register` can be defined as $\mathcal{S}_{lwwR}$ in Example 13 with the additional requirement that paths are topological orderings of visibilities

$$\mathtt{P} \in \mathcal{S}_{topR}(\mathtt{G}) \text{ iff } \mathtt{P} \in \mathcal{S}_{lwwR}(\mathtt{G}) \text{ and } \prec_{\mathtt{G}} \subseteq \leq_{\mathtt{P}}$$

In this way, $\mathcal{S}_{topR}(\mathtt{G})$ excludes e.g. the two right-most arbitrations of the equation in Figure 1a.

▶ **Definition 38** (Topological specifications). *Let* $\mathcal{S}$ *be a specification. It is* topological *if*

$$\forall \mathtt{G}, \mathcal{E}, \ell.\ \mathcal{S}(\mathtt{G}^{\ell}_{\mathcal{E}}) = \mathtt{ext}(\mathtt{pre}(\mathcal{S}(\mathtt{G}^{\ell}_{\mathcal{E}}), \mathtt{f}), \mathtt{f})$$

A *topological ps-morphism* (along ℓ) is a ps-morphism $\mathtt{f} : \mathcal{X}_1 \to \mathcal{X}_2$ with underlying function $(\mathcal{E}, \lambda) \to (\mathcal{E}_{\top}, \lambda[\top \mapsto \ell])$ such that $\mathcal{X}_2 = \mathtt{ext}(\mathtt{pre}(\mathcal{S}(\mathcal{X}_2), \mathtt{f}), \mathtt{f})$. The name is directly reminiscent of what are called topological RDTs in [10, 5], and in fact it similarly guarantees that arbitrations preserve the visibility order. We can thus prove another instance of Theorem 33, now concerning topological specifications.

▶ **Proposition 39.** *Let* $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ *be a functor mapping root extension morphisms into topological ps-morphisms (along the same labels). If* $\mathbb{F}$ *weakly preserves finite colimits, it induces a topological, coherent specification* $\mathbb{S}(\mathbb{F})$*.*

7.4 Interchangeability of Functors and Coherent Specifications

The connection between the construction of Theorem 32 and Theorem 33 is quite tight, and in fact induces a one-to-one correspondence between functors and coherent specifications.

▶ **Theorem 40.** *Let* $\mathcal{S}$ *be a coherent specification. Then* $\mathbb{S}(\mathbb{M}(\mathcal{S})) = \mathcal{S}$*. Conversely, let* $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ *be a functor verifying the hypothesis of Theorem 33. Then* $\mathbb{M}(\mathbb{S}(\mathbb{F})) \simeq \mathbb{F}$*.*

Proof. We first show that $\mathbb{M}(\mathbb{S}(\mathbb{F})) \simeq \mathbb{F}$. For notational convenience, we denote $\mathbb{M}(\mathbb{S}(\mathbb{F}))$ by $\mathbb{M}'$. We will show the existence of a natural isomorphism $\varphi \colon \mathbb{M}' \Rightarrow \mathbb{F}$. By definition, we have $\mathbb{M}'(\mathtt{G}) = \mathbb{S}(\mathbb{F})(\mathtt{G}) = \mathbb{F}(\mathtt{G})$, therefore we can define $\varphi_{\mathtt{G}} = \textsc{id}_{\mathbb{F}(\mathtt{G})}$. We need to prove that it is natural, which in this case amounts to showing $\mathbb{M}'(\mathtt{f}) = \mathbb{F}(\mathtt{f})$, for $\mathtt{f} \colon \mathtt{G} \to \mathtt{G}'$ in $\mathbf{PIDag}(\mathcal{L})$. This follows from $\mathbb{M}'(\mathtt{f})$ and $\mathbb{F}(\mathtt{f})$ having the same underlying function between events, namely the inclusion $(\mathcal{E}_{\mathtt{G}}, \lambda_{\mathtt{G}}) \to (\mathcal{E}_{\mathtt{G}'}, \lambda_{\mathtt{G}'})$.

Now we show that $\mathbb{S}(\mathbb{M}(\mathcal{S})) = \mathcal{S}$ for any coherent specification $\mathcal{S}$. This follows directly from the definition of $\mathbb{M}$ and $\mathbb{S}$. In fact, $\mathbb{S}(\mathbb{M}(\mathcal{S}))(\mathtt{G}) = \mathbb{M}(\mathcal{S})(\mathtt{G}) = \mathcal{S}(\mathtt{G})$. ◀

The one-to-one correspondence can be lifted to the specific classes of saturated/topological coherent specifications and to the functors of Proposition 35/Proposition 39, respectively. However, what is most relevant is the fact the interchangeability allows one to leverage the categorical machinery of the functor category for providing operators on specifications.

▶ Remark 41. Besides coherence, one of the keys of the previous correspondence is the (quite reasonable) choice of specifications that preserve isomorphisms. In general terms, whenever one needs to consider the relationship between different specifications, it is necessary to take into account how the underlying sets of events are related. This is quite easy to accomplish if we move to the functorial presentation. For example, we can say that a specification $\mathcal{S}_1$ refines a specification $\mathcal{S}_2$ if $\mathcal{S}_1(\mathtt{G}) \subseteq \mathcal{S}_2(\mathtt{G})$ for all graphs $\mathtt{G}$. However, this is a very concrete characterisation: it would be more general to check for the existence of a ps-morphism $\mathcal{S}_2(\mathtt{G}_2) \to \mathcal{S}_1(\mathtt{G}_1)$ whose underlying function $\mathtt{f} : \mathcal{E}_{\mathtt{G}_2} \to \mathcal{E}_{\mathtt{G}_1}$ is a bijection, in order to abstract from the identities of the events. In this case, a further constraint would be that $\mathtt{f}$ is preserved along the image of the morphisms in $\mathbf{PIDag}(\mathcal{L})$. These requirements boil down to the existence of a natural transformation $\mathbb{M}(\mathcal{S}_2) \to \mathbb{M}(\mathcal{S}_1)$.

8 Conclusions and Further Works

In this paper we have provided a functorial characterisation of RDT specifications. Our starting point is the denotational approach proposed in [7, 6], in which RDT specifications are associated with functions mapping visibility graphs into sets of admissible arbitrations that are also saturated and coherent, and where a preliminary functorial correspondence was proposed. In this paper we streamlined and expanded the latter result. We considered the category $\mathbf{PDag}(\mathcal{L})$ of labelled, acyclic graphs and pr-morphisms for representing visibility graphs. We equip $\mathbf{PDag}(\mathcal{L})$ with operators that model the evolution of visibility graphs and we show that the sub-category $\mathbf{PIDag}(\mathcal{L})$ of monic morphisms can be generated by the subset of root extensions via pushouts. For arbitrations, we take $\mathbf{SPath}(\mathcal{L})$, which is the category of sets of labelled, total orders and ps-morphisms. Then, we show that each coherent specification mapping isomorphic graphs into isomorphic set of paths induces a functor $\mathbb{M}(\mathcal{S}) : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$. Conversely, we prove that a functor $\mathbb{F} : \mathbf{PIDag}(\mathcal{L}) \to \mathbf{SPath}(\mathcal{L})$ that preserves finite colimits and binary pullbacks induces an coherent specification $\mathbb{S}(\mathbb{F})$. Moreover, $\mathbb{M}(\mathcal{S})$ and $\mathbb{S}(\mathbb{F})$ are shown to be inverses of each other.

With respect to the categorical results expressed in [7], besides the additional characterisation of topological specifications, the key improvement has been the proof that the coherence of specifications has a precise counterpart in terms of the weak preservation of colimits on their functorial presentations, as stated by Theorem 32 and Theorem 33. We thus removed the set-theoretical requirements occurring e.g. in [7, Section 5.3], as witnessed by the definition of *coherent* functor there. We believe that this purely functorial characterisation of RDTs, as further witnessed by Proposition 35 and Proposition 39, provides an ideal setting for the development of techniques for handling RDT composition, as briefly pointed out by the functorial characterisation of refinement between specifications. Our long term goal is to equip RDT specifications with a set of operators that enables us to specify and reason about complex RDTs compositionally, i.e., in terms of their constituent parts. We aim at providing a uniform formal treatment to the compositional approaches proposed in [1, 10, 12].

References

1 Carlos Baquero, Paulo Sérgio Almeida, Alcino Cunha, and Carla Ferreira. Composition in state-based replicated data types. *Bulletin of the EATCS*, 123, 2017.

2 Ahmed Bouajjani, Constantin Enea, and Jad Hamza. Verifying eventual consistency of optimistic replication systems. In Suresh Jagannathan and Peter Sewell, editors, *POPL 2014*, pages 285–296. ACM, 2014.

3 Sebastian Burckhardt, Alexey Gotsman, and Hongseok Yang. Understanding eventual consistency. Technical Report MSR-TR-2013-39, Microsoft Research, 2013.

4 Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types: specification, verification, optimality. In Suresh Jagannathan and Peter Sewell, editors, *POPL 2014*, pages 271–284. ACM, 2014.

5 Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. A framework for transactional consistency models with atomic visibility. In Luca Aceto and David de Frutos-Escrig, editors, *CONCUR 2015*, volume 42 of *LIPIcs*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

6 Fabio Gadducci, Hernán Melgratti, and Christian Roldán. On the semantics and implementation of replicated data types. *Science of Computer Programming*, 167:91–113, 2018.

7 Fabio Gadducci, Hernán C. Melgratti, and Christian Roldán. A denotational view of replicated data types. In Jean-Marie Jacquet and Mieke Massink, editors, *COORDINATION 2017*, volume 10319 of *LNCS*, pages 138–156. Springer, 2017.

8 Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

9 Alexey Gotsman and Sebastian Burckhardt. Consistency models with global operation sequencing and their composition. In Andréa W. Richa, editor, *DISC 2017*, volume 91 of *LIPIcs*, pages 23:1–23:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

10 Alexey Gotsman and Hongseok Yang. Composite replicated data types. In Jan Vitek, editor, *ESOP 2015*, volume 9032 of *LNCS*, pages 585–609. Springer, 2015.

11 Gowtham Kaki, Kapil Earanky, K. C. Sivaramakrishnan, and Suresh Jagannathan. Safe replication through bounded concurrency verification. In *OOPSLA 2018*, volume 2 of *PACMPL*, pages 164:1–164:27. ACM, 2018.

12 Adriaan Leijnse, Paulo Sérgio Almeida, and Carlos Baquero. Higher-order patterns in replicated data types. In *PaPoC 2019*. ACM, 2019.

13 Marc Shapiro, N. Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *SSS 2011*, volume 6976 of *LNCS*, pages 386–400. Springer, 2011.

14 K. C. Sivaramakrishnan, Gowtham Kaki, and Suresh Jagannathan. Declarative programming over eventually consistent data stores. In David Grove and Steve Blackburn, editors, *PLDI 2015*, pages 413–424. ACM, 2015.

New Results on Cutting Plane Proofs for Horn Constraint Systems

Hans Kleine Büning
Universität Paderborn, Paderborn, Germany
kbcsl@uni-paderborn.de

Piotr Wojciechowski
LDCSEE, West Virginia University, Morgantown, WV, USA
pwjociec@mix.wvu.edu

K. Subramani
LDCSEE, West Virginia University, Morgantown, WV, USA
k.subramani@mail.wvu.edu

Abstract

In this paper, we investigate properties of cutting plane based refutations for a class of integer programs called *Horn constraint systems (HCS).* Briefly, a system of linear inequalities $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ is called a Horn constraint system, if each entry in $\mathbf{A}$ belongs to the set $\{0, 1, -1\}$ and furthermore there is at most one positive entry per row. Our focus is on deriving refutations i.e., proofs of unsatisfiability of such programs using cutting planes as a proof system. We also look at several properties of these refutations. Horn constraint systems can be considered as a more general form of propositional Horn formulas, i.e., CNF formulas with at most one positive literal per clause. Cutting plane calculus (CP) is a well-known calculus for deciding the unsatisfiability of propositional CNF formulas and integer programs. Usually, CP consists of a pair of inference rules. These are called the addition rule (ADD) and the division rule (DIV). In this paper, we show that cutting plane calculus is still complete for Horn constraints when every intermediate constraint is required to be Horn. We also investigate the lengths of cutting plane proofs for Horn constraint systems.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Horn constraints, cutting planes, proof length

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.43

Funding This research was supported in part by the Air-Force Office of Scientific Research through Grant FA9550-19-1-017.

1 Introduction

This paper is concerned with the length of tree-like and Dag-like cutting plane refutations of Horn constraint systems (HCSs). HCSs are a type of polyhedral constraint system in which, each constraint is of the form $\mathbf{a} \cdot \mathbf{x} \geq b$, coefficients are limited to the set $\{0, 1, -1\}$, and each constraint has at most one variable with positive coefficient. HCSs find important applications in several problem domains [6].

A refutation of a system of constraints is a certificate that proves the infeasibility of that system. Associated with the concept of certificates is the concept of certifying algorithms. A certifying algorithm is any algorithm that, instead of simply returning **yes** or **no** to a feasibility query, provides a proof (certificate) that the returned response is correct [23]. Certifying algorithms for many combinatorial optimization problems have been studied in the literature. This is especially true for certifying algorithms that utilize properties of graphical structures [10, 17, 21].

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 43; pp. 43:1–43:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Certifying algorithms rely on the presence of short certificates, both positive and negative. In case of Horn constraint systems, a satisfying assignment serves as a positive certificate and it is clearly succinct. In this paper, we focus on negative certificates or refutations of Horn constraint systems; in particular, we focus on cutting plane based refutations. Our primary interest is the length of the refutations. This helps identify what restrictions can be placed on cutting plane refutations of HCSs, while still guaranteeing the existence of short refutations. In particular, we focus on the length of both tree-like refutations and Dag-like refutations. We also investigate how the length of refutations changes when different inference rules are used. In this paper, we use two well-known inference rules known as the ADD rule and DIV rule [5] (see Section 2). Additionally we study the complexity of finding read-once refutations of Horn constraint systems when we allow for constraints to be multiplied by bounded coefficients.

The principal contributions of this paper are as follows:

1. Cutting plane tree-like refutations using only the ADD rule do not p-simulate cutting plane tree-like refutations using both the ADD rule and DIV rule for HCSs (see Theorem 9).
2. There exist HCSs for which Tree-like refutations using the ADD and DIV rules must be exponential in the size of the input HCS (see Theorem 10).
3. Dag-like refutations using the ADD and DIV rules are polynomial in the size of the input HCS (see Theorem 11).
4. Finding read-once refutations of HCSs is **NP-hard** even when we allow for constraints to be multiplied by coefficients bounded by a fixed constant (see Theorem 17).

Additionally, we derive interesting corollaries from the above results.

The rest of this paper is organized as follows: In Section 2, we introduce the problems being studied. Section 3 provides motivation for studying this problem and describe the related work in the literature. Our results for tree-like refutations are presented in Section 4. In Section 5, we give our results for Dag-like refutations. We examine read-once refutations with restricted multiplication in Section 6. We conclude in Section 7 by summarizing our contributions, and outlining avenues for future research.

2 Statement of Problems

In this section, we define the problems under consideration.

▶ **Definition 1.** *A system of constraints $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ is said to be a Horn Constraint system (HCS) or a Horn polyhedron if*

1. *The entries in $\mathbf{A}$ belong to the set $\{0, 1, -1\}$.*
2. *Each row of $\mathbf{A}$ contains at most one positive entry.*
3. *$\mathbf{x}$ is a real valued vector.*
4. *$\mathbf{b}$ is an integral vector.*

In a constraint $\mathbf{a} \cdot \mathbf{x} \geq b_1$, b_1 is called the defining constant and in the constraint system $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$, $\mathbf{b}$ is referred to as the defining constant vector. The assumption that $\mathbf{b}$ is integral is *necessary* to maintain the *soundness* of the proof systems discussed in this paper.

We are interested in certificates of infeasibility; in particular, we are interested in cutting plane based refutations. In linear programs (systems of linear inequalities), we use the following rule:

$$ADD: \quad \frac{\sum_{i=1}^{n} a_i \cdot x_i \geq b_1 \qquad \sum_{i=1}^{n} a'_i \cdot x_i \geq b_2}{\sum_{i=1}^{n} (a_i + a'_i) \cdot x_i \geq b_1 + b_2} \tag{1}$$

We refer to Rule (1) as the **ADD rule**. This rule plays the same role as resolution in clausal formulas. It is easy to see that Rule (1) is sound in that any assignment satisfying the hypotheses **must** satisfy the consequent. Furthermore, the rule is **complete** in that if the original system is linear infeasible, then repeated application of Rule (1) will result in a contradiction of the form: $0 \geq b$, $b > 0$. The completeness of the ADD rule was established by Farkas [11], in a lemma that is famously known as Farkas' Lemma for systems of linear inequalities [27].

Farkas' lemma along with the fact that linear programs must have basic feasible solutions establishes that the linear programming problem is in the complexity class **NP** ∩ **coNP**. Farkas' lemma is one of several lemmata that consider pairs of linear systems in which exactly one element of the pair is feasible. These lemmas are collectively referred to as "Theorems of the Alternative" [25].

▶ **Definition 2.** *A* linear refutation *is a sequence of applications of the ADD rule that results in a contradiction of the form* $0 \geq b$, $b \geq 1$.

In general, applying the ADD rule to an infeasible system $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$, could result in a contradiction of the form $0 \geq b$, $b > 0$. However, in case of Horn systems (with integral defining constants), we must have $b \geq 1$ (see [6]).

When studying integer feasibility, we typically use an additional rule. This is referred to as the **DIV rule** and is described as follows:

$$DIV: \quad \frac{\sum_{i=1}^{n} a_{ij} \cdot x_i \geq b_j \qquad d \in \mathbb{Z}^{+} : \frac{a_{ij}}{d} \in \mathbb{Z},\, i = 1 \ldots n}{\sum_{i=1}^{n} \frac{a_{ij}}{d} \cdot x_i \geq \left\lceil \frac{b_j}{d} \right\rceil} \tag{2}$$

Rule (2) corresponds to dividing a constraint by a common divisor d of the left-hand coefficients and then rounding the right-hand side. Since each $\frac{a_{ij}}{d}$ is an integer this inference preserves integer solutions but does necessarily preserve linear solutions. However, for systems of Horn constraints the DIV rule preserves linear feasibility, since in Horn polyhedra, linear feasibility implies integer feasibility [6].

▶ **Definition 3.** *An* integer refutation *is a sequence of applications of the ADD and DIV rules that results in a contradiction of the form* $0 \geq b$, $b \geq 1$.

Note that for systems of Horn constraints, an integer refutation still proves linear infeasibility.

We now formally define the types of refutations discussed in this paper.

▶ **Definition 4.** *A* ***Dag-like*** *refutation is a refutation in which each constraint, can be used any number of times. This applies to constraints present in the original system and those derived as a result of previous applications of the inference rules.*

▶ **Definition 5.** *A* ***tree-like*** *refutation is a refutation in which each derived constraint, can be used at most once. However, if a derived constraint needs to be reused, then it can be re-derived.*

▶ **Definition 6.** *A* ***read-once*** *refutation is a refutation in which each constraint can be used at most once. This applies to constraints present in the original system and those derived as a result of previous applications of the inference rules.*

For both tree-like and Dag-like refutations, we are interested in the length of the refutation. We measure the length of a refutation in terms of the number of inferences.

▶ **Definition 7.** *The **length** of a refutation is the number of inferences (either the ADD rule or the DIV rule) in the refutation.*

We use $|S|$ to denote the length of proof S. Using this definition of length, we can now define the concept of p-simulation.

▶ **Definition 8.** *Let S and S' be two proof systems. S **p-simulates** S' over a set of formulas F, if there exists a polynomial $p(n)$ such that for every formula f in F, there exists a proof S_f of f under proof system S such that $|S_f| \leq p(|S'_f|)$ where S'_f is the shortest proof of f under proof system S'.*

This paper examines both tree-like refutations and Dag-like refutations using only the ADD rule as well as these types of refutations using both the ADD and DIV rules.

3 Motivation and Related Work

Horn constraint systems generalize difference constraint systems. Recall that a difference constraint is a relationship of the form: $x_i - x_j \geq b_{ij}$. A conjunction of difference constraints is called a Difference constraint system (DCS). It is well-known that a DCS is feasible if and only if it has an integral solution (as long as the vector of defining constants is integral). This is because the constraint matrix $\mathbf{A}$ of a DCS is Totally unimodular (TU) [27]. If $\mathbf{A}$ is TU and $\mathbf{b}$ is integral, then all the extreme points of the polyhedron $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ are integral. Horn constraint matrices are *not TU*; however, it is known that if $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ is feasible, then it has a minimal element, which is integral [32]. Horn constraint systems have been used as domains in abstract interpretation [2, 9]. Horn systems also find applications in declarative programming [16, 22]. The applications of Horn constraints to program verification has been discussed extensively in [3, 20]. Recently, Horn clauses have been utilized to solve the satisfiability problem for general CNF systems through MAXSAT resolution [4].

This paper is concerned with negative certificates. Assume that we are given a linear constraint system $\mathbf{P} : \mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ (not necessarily Horn). Any satisfying assignment to the system serves as a positive certificate which asserts the feasibility of $\mathbf{P}$. In order to certify the infeasibility of a linear system, we typically resort to Farkas' lemma [11]. As per Farkas' lemma, it suffices to provide a non-negative m-vector $\mathbf{y}$, such that $\mathbf{y} \cdot \mathbf{A} = \mathbf{0}$, $\mathbf{y} \cdot \mathbf{b} < \mathbf{0}$. This vector $\mathbf{y}$ is called the Farkas witness of the infeasibility of $\mathbf{P}$.

It is important to note that the absence of a Farkas witness guarantees linear feasibility but not integer feasibility. For establishing integer infeasibility, additional inference rules are required. One such inference system is the cutting plane calculus introduced by Gomory [12]. Gomory proposed cutting planes mainly as an algorithmic approach to solve integer programs and was less concerned with proofs and proof lengths. One of the first papers to use cutting planes as a propositional proof system is [8]. The connection between resolution and cutting planes is explored in [14]. In [13], it is shown that there exist tautologies (the pigeonhole principle) for which the number of resolution steps must be exponential in the size of the input. Exponential lower bounds for cutting plane proofs are detailed in [5] and [26]. It is unlikely that succinct certificates of infeasibility exist for integer programming, since this would mean that integer programming lies in the complexity class $\mathbf{NP} \cap \mathbf{coNP}$.

In this paper, we focus on deriving bounds on the lengths of cutting plane proofs in restricted cutting plane systems. It is to be noted that placing restrictions on the type or number of inferences that can be applied could cause the proof system to become **incomplete**. There are several reasons to consider restricted proof systems, viz.

1. Restricted proofs tend to be "short" (polynomial in the size of the input). For instance, read-once refutations are at most linear in the size of the input. Read-once and literal-once refutations have been discussed extensively for boolean formulas in [15] and [31]. This is in stark contrast to general resolution. In general resolution a refutation could have exponentially many steps [13].
2. In certain cases, the presence of restricted proofs can be determined in polynomial time. For instance, we have shown that the problem of read-once refutations is in **P** for difference constraints [29] and Unit Two Variable Per Inequality (UTPVI) constraints [30].

In recent work, we showed that the problem of finding read-once refutations under the ADD and DIV rules in Horn constraint systems is **NP-hard** [19]. In this paper, we extend these results by studying the lengths of tree-like and Dag-like refutations under the ADD and DIV rules. We also examine read-once refutations when we allow for limited use of multiplication.

4 Tree-like Refutations

In this section, we examine the length of tree-like refutations for HCSs.

First, we compare tree-like proofs using only the ADD rule to tree-like proofs using both the ADD and DIV rules.

▶ **Theorem 9.** *Tree-like proofs using only the ADD rule do not p-simulate tree-like proofs using both the ADD rule and DIV rule for HCSs.*

Proof. Consider the following HCS:

$$\begin{array}{rcl|rcl} -x_1 - x_2 - x_3 - \ldots - x_n & \geq & 1 & x_3 - \ldots - x_n & \geq & 0 \\ x_1 - x_2 - x_3 - \ldots - x_n & \geq & 0 & & \vdots & \\ x_2 - x_3 - \ldots - x_n & \geq & 0 & x_n & \geq & 0 \end{array} \tag{3}$$

We will show that any tree-like refutation of System 3 that uses only the ADD rule has at least $(2^n - 1)$ inferences. This will be done by induction on the number of variables. Let $\mathbf{H}_n$ be System 3.

If $n = 1$, then $\mathbf{H}_1$ consists of the constraints $-x_1 \geq 1$ and $x_1 \geq 0$. This system has the following refutations using only the ADD rule:

1. Apply the ADD rule to $-x_1 \geq 1$ and $x_1 \geq 0$ to get $0 \geq 1$.

This is a contradiction. Thus, System $\mathbf{H}_1$ has a refutation with $1 = 2^1 - 1$ inference. Note that this is the shortest refutation of System $\mathbf{H}_1$, thus any refutation of this system must use at least 1 inference as desired.

Now assume that when $n = k$ any refutation System $\mathbf{H}_k$ uses at least $(2^k - 1)$ inferences. Let us look at System $\mathbf{H}_{k+1}$. Without the constraint $x_{k+1} \geq 0$, System $\mathbf{H}_{k+1}$ can be satisfied by setting $\mathbf{x} = (0, 0, 0, \ldots, 0, -1)$. Thus, any refutation of System $\mathbf{H}_{k+1}$ must use $x_{k+1} \geq 0$.

Let R_{k+1} be a refutation of $\mathbf{H}_{k+1}$. Since the addition of constraints is associative we can assume without loss of generality that R_{k+1} consists of the following:

1. A derivation of a constraint of the form $-c \cdot x_{k+1} \geq 1$ for some constant c from System $\mathbf{H}_{k+1} \setminus \{x_{k+1} \geq 0\}$.
2. An additional c applications of the ADD rule (using the constraint $x_{k+1} \geq 0$) to derive the the constraint $0 \geq 1$.

Note that System $\mathbf{H}_{k+1} \setminus \{x_{k+1} \geq 0\}$ can be constructed from System $\mathbf{H}_k$ by adding $-x_{k+1}$ to every constraint. This means that any tree-like derivation of $-c \cdot x_{k+1} \geq 1$ for some constant c from System $\mathbf{H}_{k+1} \setminus \{x_{k+1} \geq 0\}$ corresponds to a refutation of System $\mathbf{H}_k$.

Thus, by the inductive hypothesis, this derivation must use at least $(2^k - 1)$ inferences. Since every constraint in $\mathbf{H}_{k+1} \setminus \{x_{k+1} \geq 0\}$ has a $-x_{k+1}$ term, we must have that $c \geq 2^k$ in the resultant constraint.

Thus, to derive the constraint $0 \geq 1$ an additional $c \geq 2^k$ inferences of the form "ADD $-c \cdot x_{k+1} \geq 1$ and $x_{k+1} \geq 0$ to get $-(c-1) \cdot x_{k+1} \geq 1$" are needed. Since we desire a tree-like refutation, we cannot shorten this refutation by reusing already derived constraints. Thus, any refutation of System $\mathbf{H}_{k+1}$ needs to use a total of $2^k + 2^k - 1 = 2^{k+1} - 1$ inferences.

However, System 3 has the following tree-like refutation using both the ADD and DIV rules:

1. Apply the ADD rule to $-x_1 - x_2 - x_3 - \ldots - x_n \geq 1$ and $x_1 - x_2 - x_3 - \ldots - x_n \geq 0$ to get
$-2 \cdot x_2 - 2 \cdot x_3 - \ldots - 2 \cdot x_n \geq 1$.
2. Apply the DIV rule with $d = 2$ to $-2{\cdot}x_2 - 2{\cdot}x_3 - \ldots - 2{\cdot}x_n \geq 1$ to get $-x_2 - x_3 - \ldots - x_n \geq 1$.
3. Apply the ADD rule to $-x_2 - x_3 - \ldots - x_n \geq 1$ and $x_2 - x_3 - \ldots - x_n \geq 0$ to get $-2 \cdot x_3 - \ldots - 2 \cdot x_n \geq 1$.
4. Apply the DIV rule with $d = 2$ to $-2 \cdot x_3 - \ldots - 2 \cdot x_n \geq 1$ to get $-x_3 - \ldots - x_n \geq 1$.
5. $\vdots$
6. Apply the ADD rule to $-x_n \geq 1$ and $x_n \geq 0$ to get $0 \geq 1$.

This refutation has only $(2 \cdot n - 1)$ inferences. Thus, the tree-like refutation of System 3 that uses only the ADD rule is exponentially longer than the tree-like refutation using both the ADD and DIV rules. ◀

Despite the fact that the DIV rule can result in much shorter tree-like refutations for systems of Horn constraints, there are still systems of Horn constraints with exponentially long refutations.

▶ **Theorem 10.** *For every positive integer n, there exists an HCS with n variables for which every tree-like cutting plane proof using both the ADD and DIV rules is exponential in the size of the system.*

Proof. We show this by introducing the variable x_0 to System (3). Specifically, we examine the following system of Horn constraints.

$$\begin{array}{rcl|rcl}
x_0 - x_1 - x_2 - x_3 - \ldots - x_n & \geq & 1 & x_3 - \ldots - x_n & \geq & 0 \\
x_1 - x_2 - x_3 - \ldots - x_n & \geq & 0 & & \vdots & \\
x_2 - x_3 - \ldots - x_n & \geq & 0 & x_n & \geq & 0 \\
 & & & -x_0 - x_i & \geq & 0
\end{array} \tag{4}$$

We will examine how the minimum length of a tree-like proof using both the ADD rule and DIV rule depends on the choice of x_i in the constraint $-x_0 - x_i \geq 0$.

By construction, the constraint $x_0 - x_1 - x_2 - x_3 - \ldots - x_n \geq 1$ is the only constraint in System (4) with the term x_0. Thus, any constraint derived from $x_0 - x_1 - x_2 - x_3 - \ldots - x_n \geq 1$ must have the term x_0 until x_0 is canceled from the constraint. This means that, the DIV rule cannot be applied (with $d > 1$) to any constraint derived from $x_0 - x_1 - x_2 - x_3 - \ldots - x_n \geq 1$ until x_0 is eliminated. There are two ways to eliminate x_0 from this constraint:

Type 1. Derive the constraint $-x_0 \geq 0$ from the remaining constraints and then eliminate x_0. In this case, the only constraint with $-x_0$ is the constraint $-x_0 - x_i \geq 0$. Thus, we must derive the constraint $x_i \geq 0$. To do this we need to eliminate $x_{i+1} \ldots x_n$ from $x_i - x_{i+1} - \ldots - x_n \geq 0$ using the constraints where those variables have positive coefficients.

Type 2. Eliminate it by using the constraint $-x_0 - x_i \geq 0$ and then eliminate the extra copy of x_i. Note that in this case the DIV rule cannot be applied until this extra copy is eliminated. To do this without deriving the constraint $x_i \geq 0$ (since this would make it the equivalent of a Type 1 refutation), we need to cancel the variables $x_1 \ldots x_{i-1}$.

We will refer to these as Type 1 and Type 2 tree-like refutations. We will show that any tree-like refutation of system (4) requires $\min(2 \cdot n + 2^{n-i}, 2 \cdot (n-i) + 2^i + 1)$ inferences.

First consider the case where $x_i = x_n$ in System (4). This results in the following HCS.

$$\begin{array}{rcl|rcl}
x_0 - x_1 - x_2 - x_3 - \ldots - x_n & \geq & 1 & x_3 - \ldots - x_n & \geq & 0 \\
 & & & & \vdots & \\
x_1 - x_2 - x_3 - \ldots - x_n & \geq & 0 & & & \\
x_2 - x_3 - \ldots - x_n & \geq & 0 & x_n & \geq & 0 \\
 & & & -x_0 - x_n & \geq & 0
\end{array}$$

In this case, System (4) has the following Type 1 tree-like refutation.

1. Apply the ADD rule to $x_n \geq 0$ and $-x_0 - x_n \geq 0$ to get $-x_0 \geq 0$.
2. Apply the ADD rule to $-x_0 \geq 0$ and $x_0 - x_1 - x_2 - x_3 - \ldots - x_n \geq 1$ to get $-x_1 - x_2 - x_3 - \ldots - x_n \geq 1$.

We have now derived System (3) with n variables. From Theorem 9, System (3) has a tree-like refutation of length $(2 \cdot n - 1)$. Thus, if $x_i = x_n$, System (4) has a refutation of length $(2 \cdot n + 1)$ as desired.

However, consider the case where $x_i = x_1$ in System 4. This results in the following HCS.

$$\begin{array}{rcl|rcl}
x_0 - x_1 - x_2 - x_3 - \ldots - x_n & \geq & 1 & x_3 - \ldots - x_n & \geq & 0 \\
 & & & & \vdots & \\
x_1 - x_2 - x_3 - \ldots - x_n & \geq & 0 & & & \\
x_2 - x_3 - \ldots - x_n & \geq & 0 & x_n & \geq & 0 \\
 & & & -x_0 - x_1 & \geq & 0
\end{array}$$

In this case, System (4) has the following Type 2 tree-like refutation.

1. Apply the ADD rule to $-x_0 - x_1 \geq 0$ and $x_0 - x_1 - x_2 - x_3 \ldots x_n \geq 1$ to get $-2 \cdot x_1 - x_2 - x_3 \ldots - x_n \geq 1$.
2. Apply the ADD rule to $x_1 - x_2 - x_3 - \ldots - x_n \geq 0$ and $-2 \cdot x_1 - x_2 - x_3 \ldots - x_n \geq 1$ to get $-x_1 - 2 \cdot x_2 - 2 \cdot x_3 - \ldots - 2 \cdot x_n \geq 1$.
3. Apply the ADD rule to $x_1 - x_2 - x_3 - \ldots - x_n \geq 0$ and $-x_1 - 2 \cdot x_2 - 2 \cdot x_3 \ldots - 2 \cdot x_n \geq 1$ to get $-3 \cdot x_2 - 3 \cdot x_3 - \ldots - 3 \cdot x_n \geq 1$.
4. Apply the DIV rule with $d = 3$ to $-3 \cdot x_2 - 3 \cdot x_3 - \ldots - 3 \cdot x_n \geq 1$ to get $-x_2 - x_3 - \ldots - x_n \geq 1$.

This results in System (3) with $(n-1)$ variables. This means that completing the refutation will take an additional $(2 \cdot n - 3)$ inferences. Thus, in this case, System (4) has a refutation of length $(2 \cdot n + 1)$ as desired.

Now we consider System (4). As stated previously, any tree-like refutation must be Type 1 or Type 2. Thus, we need to calculate the minimum number of inferences used by each type of refutation.

First we find the minimum length of a Type 1 tree-like refutation. In the general case for System (4), this refutation has the following form:

1. Apply the ADD rule to

$$x_i - x_{i+1} - \ldots - x_n \geq 0 \text{ and } x_{i+1} - x_{i+2} \ldots - x_n \geq 0$$

to get

$$x_i - 2 \cdot x_{i+2} - \ldots - 2 \cdot x_n \geq 0.$$

2. Apply the ADD rule to $x_i - 2 \cdot x_{i+2} - \ldots - 2 \cdot x_n \geq 0$ and 2 copies of $x_{i+2} - x_{i+3} \ldots - x_n \geq 0$ to get
$$x_i - 4 \cdot x_{i+3} - \ldots - 4 \cdot x_n \geq 0.$$
3. For each $r = 3 \ldots (n-i-1)$, apply the ADD rule to
$$x_i - 2^{r-1} \cdot x_{i+r} - \ldots - 2^{r-1} \cdot x_n \geq 0$$
and 2^{r-1} copies of $x_{+r} - x_{i+r+1} \ldots - x_n \geq 0$ to get
$$x_i - 2^r \cdot x_{i+r+1} - \ldots - 2^r \cdot x_n \geq 0.$$
4. Apply the ADD rule to $x_i - 2^{n-i-1} \cdot x_n$ and 2^{n-i-1} copies of $x_n \geq 0$ to get $x_i \geq 0$.
5. Apply the ADD rule to $x_i \geq 0$ and $-x_0 - x_i \geq 0$ to get $-x_0 \geq 0$.
6. Apply the ADD rule to $-x_0 \geq 0$ and $x_0 - x_1 - x_2 - x_3 - \ldots - x_n \geq 1$ to get
$$-x_1 - x_2 - x_3 - \ldots - x_n \geq 1.$$

This takes a minimum of $(2^{n-i} + 1)$ inferences. We have now derived System (3) with n variables. From Theorem 9, System (3) has a tree-like refutation of length $(2 \cdot n - 1)$. Thus, the minimum length of a Type 1 tree-like refutation of System (4) is $(2 \cdot n + 2^{n-i})$ as desired.

Now we find the minimum length of a Type 2 tree-like refutation. In the general case for System (4), this refutation has the following form:

1. Apply the ADD rule to
$$-x_0 - x_i \geq 0 \text{ and } x_0 - x_1 - x_2 - x_3 \ldots - x_n \geq 1$$
to get
$$-x_1 - x_2 - \ldots - 2 \cdot x_i - \ldots - x_n \geq 1$$
2. Apply the ADD rule to
$$x_1 - x_2 - x_3 - \ldots - x_n \geq 0 \text{ and } -x_1 - x_2 - \ldots - 2 \cdot x_i - \ldots - x_n \geq 1$$
to get
$$-2 \cdot x_2 - 2 \cdot x_3 - \ldots - 3 \cdot x_i - \ldots - 2 \cdot x_n \geq 1.$$
3. Apply the ADD rule to $-2 \cdot x_2 - 2 \cdot x_3 - \ldots - 3 \cdot x_i - \ldots - 2 \cdot x_n \geq 1$ and 2 copies of $x_2 - x_3 - \ldots - x_n \geq 0$ to get
$$-4 \cdot x_3 - 4 \cdot x_4 - \ldots - 5 \cdot x_i - \ldots - 4 \cdot x_n \geq 1.$$
4. For each $r = 3 \ldots i - 1$, apply the ADD rule to
$$-2^{r-1} \cdot x_r - 2^{r-1} \cdot x_{r+1} - \ldots - (2^{r-1} + 1) \cdot x_i - \ldots - 2^{r-1} \cdot x_n \geq 1$$
and 2^{r-1} copies of $x_r - x_{r+1} - \ldots - x_n \geq 0$ to get
$$-2^r \cdot x_{r+1} - 2^r \cdot x_{r+2} - \ldots - (2^r + 1) \cdot x_i - \ldots - 2^r \cdot x_n \geq 1.$$
5. Apply the ADD rule to $-(2^{i-1} + 1) \cdot x_i - 2^{i-1} \cdot x_{i+1} - \ldots - 2^{i-1} \cdot x_n \geq 1$ and $(2^{i-1} + 1)$ copies of $x_i - x_{i+1} \ldots - x_n \geq 0$ to get
$$-(2^i + 1) \cdot x_{i+1} - (2^i + 1) \cdot x_{i+2} - \ldots - (2^i + 1) \cdot x_n \geq 1.$$

6. Apply the DIV rule with $d = (2^i + 1)$ to

$$-(2^i + 1) \cdot x_{i+1} - (2^i + 1) \cdot x_{i+2} - \ldots - (2^i + 1) \cdot x_n \geq 1$$

to get

$$-x_{i+1} - x_{i+2} - \ldots - x_n \geq 1.$$

This takes a minimum of $(2^i + 2)$ inferences. We have now derived System (3) with $(n - i)$ variables. This means that completing the refutation will take an additional $(2 \cdot n - i - 1)$ inferences. Thus, the minimum length of a Type 2 tree-like refutation of System (4) is $(2 \cdot (n - i) + 2^i + 1)$ as desired.

Thus, when $x_i = x_{\frac{n}{2}}$ any tree-like refutation of System 4 must have a length of at least $(n + 2^{\frac{n}{2}} + 1)$.

$$\begin{array}{rcl|rcl}
x_0 - x_1 - x_2 - x_3 - \ldots - x_n & \geq & 1 & x_3 - \ldots - x_n & \geq & 0 \\
 & & & \vdots & & \\
x_1 - x_2 - x_3 - \ldots - x_n & \geq & 0 & & & \\
x_2 - x_3 - \ldots - x_n & \geq & 0 & x_n & \geq & 0 \\
 & & & -x_0 - x_{\frac{n}{2}} & \geq & 0
\end{array}$$

Thus, any tree-like refutation of this system that uses both the ADD rule and the DIV rule must be exponential in the size of the system. ◀

5 Dag-like Refutations

In this section, we examine the lengths of Dag-like refutations of systems of Horn constraints.

Unlike tree-like refutations, Dag-like refutations of HCSs are guaranteed to be polynomially sized.

▶ **Theorem 11.** *Dag-like cutting plane proofs of Horn constraint systems are polynomial in the size of the constraint system even when restricted to using only the ADD rule.*

Proof. Let $\mathbf{H}$ be an unsatisfiable system of Horn constraints with m constraints over n variables. If $\mathbf{H}$ has no positive absolute constraints (constraints of the form $x_i \geq c$), then let $\mathbf{D} \subseteq \mathbf{H}$ be the set of difference constraints in $\mathbf{H}$. If $\mathbf{x}$ is a assignment to the variables in $\mathbf{H}$ that satisfies every constraint in $\mathbf{D}$, then for some positive constant M, $(\mathbf{x} - M \cdot \mathbf{1})$ is a satisfying assignment to $\mathbf{H}$. Thus, $\mathbf{D}$ must be unsatisfiable. It is easy in this case $\mathbf{H}$ has a refutation of length at most m since $\mathbf{D}$ is an infeasible system of difference constraints.

If $\mathbf{H}$ has a positive absolute constraint $x_i \geq c$, then we can construct the system $\mathbf{H}'$ by removing the constraint $x_i \geq c$ and summing it with every constraint with the literal $-x_i$. This process is repeated, until a feasible system is derived or an infeasible system of difference constraints is constructed.

Note that if an infeasible system of difference constraints is constructed, then the system has a read-once refutation using only the ADD rule. This is a linearly sized proof of infeasibility. To obtain this system, we eliminated at most $(n - 1)$ constraints of the form $x_i \geq c$. Each of these eliminations took at most m applications of the ADD rule. Since the refutation of the resultant DCS is read-once, it cannot use more than m inferences. Thus, any refutation discovered by this process has length at most $m \cdot n$. ◀

Note that the refutation generated this way has the following properties:

1. Every intermediate constraint is a Horn constraint.
2. Only the ADD rule is used.

Thus, we have the following corollaries:

▶ **Corollary 12.** *Every infeasible system of Horn constraints has a refutation where every intermediate constraint is Horn.*

Proof. This follows immediately from the fact that in the refutation generated by Theorem 11, every intermediate constraint is Horn. ◀

▶ **Corollary 13.** *If a Dag-like refutation of length n exists for an unsatisfiable Horn constraint system H, then there exists a polynomial p such that there exists Dag-like refutation of length $p(n)$, even when all intermediate constraints are Horn.*

Proof. Let D be a Dag-like refutation of H of length n. Let $H_D \subseteq H$ be the set of constraints in H used by D. Thus, H_D is an infeasible HCS. By Theorem 11, H_D has a polynomially sized Dag-like refutation D' of length n' where every intermediate constraint is Horn. Thus, there exists a polynomial p such that $n' \leq p(|H_D|) \leq p(n)$ since $|H_D| \leq n$. ◀

▶ **Corollary 14.** *Dag-like refutations using only the ADD rule p-simulate Dag-like refutations using both the ADD rule and DIV rule for HCSs.*

Proof. Let H be an arbitrary HCS. Let D be the shortest Dag-like refutation of H that uses both the ADD rule and the DIV rule, and let n be the length of D. Let $H_D \subseteq H$ be the set of constraints in H used by D. Thus, H_D is an infeasible HCS. By Theorem 11, H_D has a polynomially sized Dag-like refutation D' of length n' using only the ADD rule. Thus, there exists a polynomial p such that $n' \leq p(|H_D|) \leq p(n)$ since $|H_D| \leq n$. ◀

6 Restricted Read-once Refutations

In this section, we examine the complexity of finding read-once refutations of HCSs when we allow for the multiplication of constraints by bounded coefficients. To accomplish this, we introduce an inference rule known as the **MUL rule**. This rule is described as follows:

$$MUL : \quad \frac{\sum_{i=1}^{n} a_{ij} \cdot x_i \geq b_j \qquad\qquad c_j \in \mathbb{Z}^+}{\sum_{i=1}^{n} c_j \cdot a_{ij} \cdot x_i \geq c_j \cdot b_j} \tag{5}$$

Rule (5) corresponds to multiplying a constraint by a positive integer multiplier. Note that, with unrestricted use of the ADD and MUL rules, any infeasible system of Horn constraints has a read-once refutation. This refutation is constructed as follows:

1. Let $\mathbf{H}$ be an infeasible HCS, and let T be a tree-like refutation of H that uses only the ADD rule.
2. For each constraint $l_j \in \mathbf{H}$, let c_j be the number of times l_j is used in T. If $c_j > 0$, apply the MUL rule, with multiplier c_j to l_j.
3. Use the ADD rule to sum together all the constraints generated with the MUL rule.

Thus, we examine the problem of finding read-once refutations in HCSs when we only allow for restricted use of the MUL rule.

▶ **Definition 15.** *An r-restricted read-once refutation using the ADD and MUL rules of an HCS $\mathbf{H}$, is a read-once refutation of $\mathbf{H}$ such that:*

1. *The MUL rule is only applied with with multiplier $c \leq r$.*
2. *The MUL rule is only applied to constraints in $\mathbf{H}$.*

First we show that, for any fixed constant r, the problem of finding an r-restricted read-once refutation using the ADD and MUL rules is **NP-hard**. To accomplish this, we utilize a reduction from the set packing problem.

▶ **Definition 16.** *The **set packing** problem is the following: Given a set S, m subsets $S_1, \ldots, S_m$ of S, and an integer k, does $\{S_1, \ldots, S_m\}$ contain k mutually disjoint sets.*

This problem is known to be **NP-complete** [18].

▶ **Theorem 17.** *Let r be any positive integer. Finding r-restricted read-once refutations using the ADD and MUL rules is **NP-hard** for HCSs.*

Proof. Consider an instance of the set packing problem and let h be the integer such that $2^h \leq r < 2^{h+1}$. We construct the system of Horn constraints $\mathbf{H}$ as follows:

1. For each $x_i \in S$:
   a. Create the variables x_i and w_i.
   b. For each $g = 1 \ldots h$, create the variables $y_{i,(2 \cdot g-1)}$ and $y_{i,(2 \cdot g)}$. Also create the constraints $y_{i,(2 \cdot g-1)} - y_{i,(2 \cdot g+1)} - y_{i,(2 \cdot g+2)} \geq 0$ and $y_{i,(2 \cdot g)} - y_{i,(2 \cdot g+1)} - y_{i,(2 \cdot g+2)} \geq 0$.
   c. Create the constraints $x_i - y_{i,1} - y_{i,2} \geq 0$, $y_{i,(2 \cdot h-1)} - w_i \geq 0$, $y_{i,(2 \cdot h)} - w_i \geq 0$, and $w_i \geq 0$.
2. For $j = 1 \ldots k$, create the variable v_j.
3. For each subset S_l, $l = 1 \ldots m$, and each $j = 1 \ldots k$ create the constraints $v_j - \sum_{x_i \in S_l} x_i \geq 0$.
4. Finally create the constraint $-v_1 - \ldots - v_k \geq 1$.

We will show that $\mathbf{H}$ has an r-restricted read-once refutation using the ADD and MUL rules if and only if the sets S_1, …, S_m have a packing of size k. First we assume that $\mathbf{H}$ has a r-restricted read-once refutation R using the ADD and MUL rules.

Consider the constraint $x_i - y_{i,1} - y_{i,2} \geq 0$. Assume that R applies the MUL rule to this constraint with multiplier c_i. This results in the constraint $c_i \cdot x_i - c_i \cdot y_{i,1} - c_i \cdot y_{i,2} \geq 0$ To eliminate the variables $y_{i,1}$ and $y_{i,2}$, R must also do the following:

1. Apply the MUL rule with multiplier c_i to each the constraints $y_{i,1} - y_{i,3} - y_{i,4} \geq 0$ and $y_{i,2} - y_{i,3} - y_{i,4} \geq 0$. Then apply the ADD rule to generate the constraint $c_i \cdot x_i - 2 \cdot c_i \cdot y_{i,3} - 2 \cdot c_i \cdot y_{i,4} \geq 0$.
2. Apply the MUL rule with multiplier $2 \cdot c_i$ to each the constraints $y_{i,3} - y_{i,5} - y_{i,6} \geq 0$ and $y_{i,4} - y_{i,5} - y_{i,6} \geq 0$. Then apply the ADD rule to generate the constraint $c_i \cdot x_i - 4 \cdot c_i \cdot y_{i,5} - 4 \cdot c_i \cdot y_{i,6} \geq 0$.
3. Apply the MUL rule with multiplier $4 \cdot c_i$ to each the constraints $y_{i,5} - y_{i,7} - y_{i,8} \geq 0$ and $y_{i,6} - y_{i,7} - y_{i,8} \geq 0$. Then apply the ADD rule to generate the constraint $c_i \cdot x_i - 8 \cdot c_i \cdot y_{i,7} - 8 \cdot c_i \cdot y_{i,8} \geq 0$.
4. Apply the MUL rule with multiplier $2^{h-1} \cdot c_i$ to each the constraints $y_{i,(2 \cdot h-1)} - w_i \geq 0$ and $y_{i,(2 \cdot h)} - w_i \geq 0$. Then apply the ADD rule to generate the constraint $c_i \cdot x_i - 2^h \cdot c_i \cdot w_i \geq 0$.
5. Apply the MUL rule with multiplier $2^h \cdot c_i$ to the constraint $w_i \geq 0$. Then apply the ADD rule to generate the constraint $c_i \cdot x_i \geq 0$.

Since R is an r-restricted read-once refutation, we have that $2^h \cdot c_i \leq r < 2 \cdot 2^h$. Thus, $c_i = 1$.

By construction, $x_i - y_{i,1} - y_{i,2} \geq 0$ is the only constraint in $\mathbf{H}$ where x_i has positive coefficient. Thus, for each x_i, R can only use one constraint where x_i has negative coefficient. Otherwise, R will be unable to cancel every instance of $-x_i$.

Note that, by construction, the only constraint in $\mathbf{H}$ with positive defining constant is $-v_1 - \ldots - v_k \geq 1$. Thus, this constraint must by used in R. For each v_i, R must use a constraint corresponding to one of the sets $S_1, \ldots, S_m$. Recall, that x_i can be used by at most one of these constraints, thus the sets chosen for each v_j must be mutually disjoint and no set can be chosen multiple times. This can only happen if the sets $S_1, \ldots, S_m$ have a packing of size k.

Now assume that the sets $S_1, \ldots, S_m$ have a packing of size k. Assume without loss of generality that this packing is the sets $S_1, \ldots, S_k$. H has the following r-restricted read-once refutation using the ADD and MUL rules.

1. Start with the constraint $-v_1 - \ldots - v_k \geq 1$.
2. For each v_j, apply the ADD rule to the constraint $v_j - \sum_{x_i \in S_j} x_i \geq 0$ and the result of the previous application of the ADD rule.
3. Let L be the constraint derived through this process. Since the sets $S_1, \ldots, S_k$ are mutually disjoint, every variable x_i in L has coefficient -1.
4. For each x_i in L, derive the constraint $x_i \geq 0$, using the method detailed previously. Then use the constraint $x_i \geq 0$ to eliminate $-x_i$ from L. ◀

Note that the construction used in Theorem 17 assumes that r is a fixed constant. Let n be the number of variables in the HCS $\mathbf{H}$ constructed in the proof of Theorem 17. By construction, we have that $n = k + (2 \cdot \log_2 r + 2) \cdot |S|$. We can assume without loss of generality that $k \leq |S|$ (we cannot pack more than k non-empty subsets of S into S). Thus, $n \leq |S| \cdot (2 \cdot \log_2 r + 3)$. Let c be an arbitrary positive integer. If we choose $r = 2^{\frac{|S|^{c-1}-3}{2}}$, then $|S|^{c-1} = 2 \cdot (\log_2 r) + 3$ and $n^{c-1} \leq (2 \cdot (\log_2 r) + 3)^c$. This means that $\log_2 r$ is in $O(n^{1-\frac{1}{c}})$. This leads to the following corollary of Theorem 17.

▶ **Corollary 18.** *Let c be an arbitrary positive integer. Finding r-restricted read-once refutations using the ADD and MUL rules is **NP-hard** for HCSs, even when r is $O\left(2^{(n^{1-\frac{1}{c}})}\right)$.*

7 Conclusion

In this paper, we studied refutability in Horn constraint systems under various cutting plane based proof systems. Horn constraint systems generalize difference constraint systems and find applications in a number of domains, especially program verification. We looked at both tree-like and Dag-like refutations. Both these proof systems are complete for Horn constraint systems (assuming that the defining constants are integers). For both types of refutations we looked at refutations using only the ADD rule as well as refutations using both the ADD and DIV rules. We showed there exist HCSs with exponentially sized tree-like refutations even when both the ADD and DIV rules are allowed. We also showed that every HCS has a polynomially sized Dag-like refutation even when restricted to only the ADD rule. It follows that cutting plane Dag-like refutations using only the ADD rule p-simulate cutting plane Dag-like refutations using the ADD rule and DIV rule for HCSs. Additionally, we established that if a cutting plane refutation of length n exists for an unsatisfiable Horn constraint system, then there exists a cutting plane proof of length $p(n)$, even when all intermediate constraints are Horn. We also showed that, when we allow for constraints to be multiplied by bounded coefficients, the problem of finding a read-once refutation of an HCS is **NP-complete**. Our results are important because they are the first step towards the design of efficient procedures in constraint solvers [7, 24, 28].

From our perspective, the following problems are worth pursuing:

1. Can we find the shortest tree-like or Dag-like proofs of an unsatisfiable Horn constraint system, if only the ADD rule is permitted?
2. Can we find the shortest tree-like or Dag-like proofs of an unsatisfiable Horn constraint system, if both the ADD and the DIV rules are permitted?

It is worth noting that in the case of Horn clauses the problem of finding the shortest resolution proof is **NP-hard** [1]. Even worse, the problem is hard to linearly approximate [1]. However, these negative results do not directly apply to tree-like (or Dag-like) proofs in Horn constraint systems.

References

1 M. Alekhnovich, S. Buss, S. Moran, and T. Pitassi. Minimum Propositional Proof Length is NP-Hard to Linearly Approximate. In *Mathematical Foundations of Computer Science (MFCS)*, pages 176–184. Springer-Verlag, 1998. Lecture Notes in Computer Science.

2 Alexey Bakhirkin and David Monniaux. Combining Forward and Backward Abstract Interpretation of Horn Clauses. In *Static Analysis - 24th International Symposium, SAS 2017, New York, NY, USA, August 30 - September 1, 2017, Proceedings*, pages 23–45, 2017.

3 Nikolaj Bjørner, Arie Gurfinkel, Kenneth L. McMillan, and Andrey Rybalchenko. Horn Clause Solvers for Program Verification. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, pages 24–51, 2015.

4 Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, Joao Marques-Silva, and Antonio Morgado. MaxSAT Resolution With the Dual Rail Encoding. In *AAAI Conference on Artificial Intelligence*, 2018. URL: `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16782/16235`.

5 Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. Lower Bounds for Cutting Planes Proofs with Small Coefficients. *J. Symb. Log.*, 62(3):708–728, 1997.

6 R. Chandrasekaran and K. Subramani. A combinatorial algorithm for Horn programs. *Discrete Optimization*, 10:85–101, 2013.

7 Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Computing Small Unsatisfiable Cores in Satisfiability Modulo Theories. *J. Artif. Intell. Res. (JAIR)*, 40:701–728, 2011.

8 W. Cook, C. R. Coullard, and Gy. Turan. On the complexity of Cutting-Plane Proofs. *Discrete Applied Mathematics*, 18:25–38, 1987.

9 Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*, pages 238–252, 1977.

10 Marcel Dhiflaoui, Stefan Funke, Carsten Kwappik, Kurt Mehlhorn, Michael Seel, Elmar Schömer, Ralph Schulte, and Dennis Weber. Certifying and repairing solutions to large LPs how good are LP-solvers? In *SODA*, pages 255–256, 2003.

11 Gyula Farkas. Über die Theorie der Einfachen Ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124(124):1–27, 1902.

12 R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

13 A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

14 John N. Hooker. Generalized Resolution and Cutting Planes. *Annals of Operations Research*, 12(1-4):217–239, 1988.

15 K. Iwama and E. Miyano. Intractability of Read-Once Resolution. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory (SCTC '95)*, pages 29–36, Los Alamitos, CA, USA, June 1995. IEEE Computer Society Press.

16 Joxan Jaffar and Michael Maher. Constraint Logic Programming: A Survey. *The Journal of Logic Programming*, s 19–20:503–581, October 1994. doi:10.1016/0743-1066(94)90033-7.

17 Haim Kaplan and Yahav Nussbaum. Certifying algorithms for recognizing proper circular-arc graphs and unit circular-arc graphs. *Discrete Applied Mathematics*, 157(15):3216–3230, 2009.

18 Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.

19 Hans Kleine Büning, Piotr J. Wojciechowski, and K. Subramani. On the application of restricted cutting plane systems to Horn constraint systems. In *The 12th International Symposium on Frontiers of Combining Systems, London, United Kingdom,, September 4-6, 2019, Proceedings*, pages 149–164, 2019.

20 Anvesh Komuravelli, Nikolaj Bjørner, Arie Gurfinkel, and Kenneth L. McMillan. Compositional Verification of Procedural Programs using Horn Clauses over Integers and Arrays. In *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, pages 89–96, 2015.

21 Dieter Kratsch, Ross M. McConnell, Kurt Mehlhorn, and Jeremy Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. In *SODA*, pages 158–167, 2003.

22 Kung-Kiu Lau and Mario Ornaghi. Specifying Compositional Units for Correct Program Development in Computational Logic. In *Program Development in Computational Logic: A Decade of Research Advances in Logic-Based Program Development*, pages 1–29. Springer, 2004.

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

24 Microsoft Research. *Z3: An efficient SMT solver.* URL: http://research.microsoft.com/projects/z3/.

25 G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, New York, 1999.

26 Pavel Pudlák. Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations. *J. Symb. Log.*, 62(3):981–998, 1997. doi:10.2307/2275583.

27 A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1987.

28 SRI International. *Yices: An SMT solver.* URL: http://yices.csl.sri.com/.

29 K. Subramani. Optimal Length Resolution Refutations of Difference Constraint Systems. *Journal of Automated Reasoning (JAR)*, 43(2):121–137, 2009.

30 K. Subramani and Piotr Wojciechowki. A Polynomial Time Algorithm for Read-Once Certification of Linear Infeasibility in UTVPI Constraints. *Algorithmica*, 81(7):2765–2794, 2019.

31 Stefan Szeider. NP-completeness of refutability by literal-once resolution. In *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, pages 168–181, 2001.

32 A.F. Veinott and G.B. Dantzig. Integral Extreme points. *SIAM Review*, 10:371–372, 1968.

The Tree-Generative Capacity of Combinatory Categorial Grammars

Marco Kuhlmann
Dept. of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden
marco.kuhlmann@liu.se

Andreas Maletti
Institute for Computer Science, Universität Leipzig, P.O. box 100 920, D-04009 Leipzig, Germany
maletti@informatik.uni-leipzig.de

Lena Katharina Schiffer
Institute for Computer Science, Universität Leipzig, P.O. box 100 920, D-04009 Leipzig, Germany
schiffer@informatik.uni-leipzig.de

Abstract

The generative capacity of combinatory categorial grammars as acceptors of tree languages is investigated. It is demonstrated that the such obtained tree languages can also be generated by simple monadic context-free tree grammars. However, the subclass of pure combinatory categorial grammars cannot even accept all regular tree languages. Additionally, the tree languages accepted by combinatory categorial grammars with limited rule degrees are characterized: If only application rules are allowed, then they can accept only a proper subset of the regular tree languages, whereas they can accept exactly the regular tree languages once first degree composition rules are permitted.

2012 ACM Subject Classification Theory of computation; Theory of computation → Formal languages and automata theory; Theory of computation → Tree languages; Theory of computation → Grammars and context-free languages; Computing methodologies; Computing methodologies → Artificial intelligence; Computing methodologies → Natural language processing

Keywords and phrases Combinatory Categorial Grammar, Regular Tree Language, Linear Context-free Tree Language

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.44

Funding *Marco Kuhlmann*: Supported by the Centre for Industrial IT (CENIIT), grant 15.02.
Lena Katharina Schiffer: Supported by the German Research Foundation (DFG) Research Training Group GRK 1763 "Quantitative Logics and Automata".

1 Introduction

Categorial grammars [5] were introduced alongside the phrase-structure grammars (regular, context-free, context-sensitive grammars, etc.) of the Chomsky hierarchy [6] inspired by classical notions from proof theory [1, 3]. Combinatory Categorial Grammar (CCG) [23, 24] is an extension following the approach of combinatory logic [22, 7]. CCG received considerable attention in theoretical computer science culminating in the proofs of its mild context-sensitivity, which in particular, requires efficient parsing [27], as well as its equivalence to several other established grammar formalisms [28]. It has since become a widely applied formalism in computational linguistics [18, 17].

The basis for CCG is provided by a lexicon and a rule system. The lexicon assigns syntactic categories to the symbols of the input and the rule system describes how adjoining categories can be combined to eventually obtain a (binary) derivation tree. The mentioned equivalence result due to Vijay-Shanker and Weir [28] shows that CCG, Tree-Adjoining Grammar (TAG) [12] as well as linear indexed grammars [11] are equivalent in expressive power, which establishes that they generate the same string languages. However, the used

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 44; pp. 44:1–44:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

construction depends on the ability to restrict the combination rules and to include entries for the empty word in the lexicon. Modern variants of CCG disfavor rule restrictions and the obtained *pure* CCG are strictly less expressive than TAG [14] unless unbounded generalized composition rules are permitted, in which case they are strictly more expressive than TAG [26]. Indeed, CCG with unbounded composition rules, rule restrictions as well as ε-entries in the lexicon are Turing-complete [15].

The mentioned studies examine the string (or weak) generative capacity of CCG, but already [26] asks for the tree (or strong) generative capacity or, more specifically, the expressiveness of the tree languages of CCG derivation trees [10]. Koller and Kuhlmann [13] show that CCG and TAG generate incomparable classes of *dependency trees*. In this contribution, we answer the original question and characterize the tree languages accepted by CCGs, and relate them to the standard notions of regular [9, 10] and context-free tree languages [19, 20]. A tree language $\mathcal{F}$ is *accepted* by a CCG G if $\mathcal{F}$ is obtained as a relabeling of the derivation tree language of G. Our work therefore is similar in spirit to that of Tiede [25], who studied the strong generative capacity of Lambek-style categorial grammars [16].

In the variant of CCG we investigate, the rule system is finite and includes only application and composition operators (i.e. rules based on the **B**-combinator of combinatory logic [8]). In general, we allow rule restrictions that further constrain the categories the rules can be applied to. Notice that our results concern only binary trees since the derivation trees of CCGs are binary. Our main result is that the tree languages accepted by CCGs can also be generated by simple monadic context-free tree grammars (Theorem 20). For CCG without rule restrictions this inclusion is proper since not even all regular tree languages [10] are accepted by these CCGs (Theorem 22). In addition, we show that CCGs without composition operations, which are weakly equivalent to (ε-free) context-free grammars, generate a strict subclass of the regular tree languages that does not even include all local tree languages (Theorem 9). Finally, if we limit the permitted composition operators to first degree, then exactly the regular tree languages are accepted (Theorem 14).

2 Preliminaries

We denote the set of nonnegative integers by $\mathbb{N}$ and let $[k] = \{i \in \mathbb{N} \mid 1 \leq i \leq k\}$ for every $k \in \mathbb{N}$. The power-set (i.e. set of all subsets) of a set A is $\mathcal{P}(A) = \{A' \mid A' \subseteq A\}$, and $\mathcal{P}_+(A) = \mathcal{P}(A) \setminus \{\emptyset\}$ contains all nonempty subsets. As usual, an alphabet is a finite set of symbols. The monoid $(\Sigma^*, \cdot, \varepsilon)$ consists of all strings (i.e. sequences) over a set Σ together with concatenation $\cdot$ and the empty string ε. We often write concatenation by juxtaposition. The length of a string $w \in \Sigma^*$ (i.e. the number of components in the sequence) is denoted by $|w|$. Any set $\mathcal{L} \subseteq \Sigma^*$ is a language, and the languages form a monoid $(\mathcal{P}(\Sigma^*), \cdot, \{\varepsilon\})$ with concatenation lifted to languages by $\mathcal{L} \cdot \mathcal{L}' = \{w \cdot w' \mid w \in \mathcal{L},\, w' \in \mathcal{L}'\}$. Every mapping $f \colon \Sigma \to \Delta^*$ [respectively, $f \colon \Sigma \to \mathcal{P}(\Delta^*)$] extends uniquely to a monoid homomorphism $f' \colon \Sigma^* \to \Delta^*$ [respectively, $f' \colon \Sigma^* \to \mathcal{P}(\Delta^*)$]. We will not distinguish the mapping f and its induced homomorphism f', but rather use f for both.

Given two sets A and A', a relation from A to A' is a subset $\rho \subseteq A \times A'$. The inverse of ρ is $\rho^{-1} = \{(a', a) \mid (a, a') \in \rho\}$, and for every $B \subseteq A$, we let $\rho(B) = \{a' \mid \exists b \in B \colon (b, a') \in \rho\}$. The relation $\rho \subseteq A \times A'$ can also be understood as a mapping $\widehat{\rho} \colon A \to \mathcal{P}(A')$ with $\widehat{\rho}(a) = \rho(\{a\})$ for all $a \in A$. We will not distinguish these two representations.

We build binary trees over the set Σ_2 of binary internal symbols, the alphabet Σ_1 of unary internal symbols, and the alphabet Σ_0 of leaf symbols.[1] Formally, the set $T_{\Sigma_2,\Sigma_1}(\Sigma_0)$ of binary (Σ_2, Σ_1)-trees indexed by Σ_0 is the smallest set T such that (i) $a \in T$ for all $a \in \Sigma_0$,

[1] We explicitly allow an infinite set of internal binary symbols.

(ii) $n(t) \in T$ for all $n \in \Sigma_1$ and $t \in T$, and (iii) $c(t_1, t_2) \in T$ for all $c \in \Sigma_2$ and $t_1, t_2 \in T$. We use graphical representations of trees to increase the readability. Every subset $\mathcal{F} \subseteq T_{\Sigma_2,\Sigma_1}(\Sigma_0)$ is a tree language. The mapping $\mathrm{pos}\colon T_{\Sigma_2,\Sigma_1}(\Sigma_0) \to \mathcal{P}_+(\{1,2\}^*)$ assigning positions to a tree is defined by (i) $\mathrm{pos}(a) = \{\varepsilon\}$ for all $a \in \Sigma_0$, (ii) $\mathrm{pos}(n(t)) = \{\varepsilon\} \cup \{1 \cdot w \mid w \in \mathrm{pos}(t)\}$ for all $n \in \Sigma_1$ and $t \in T_{\Sigma_2,\Sigma_1}(\Sigma_0)$, and (iii) for all $c \in \Sigma_2$ and $t_1, t_2 \in T_{\Sigma_2,\Sigma_1}(\Sigma_0)$,

$$\mathrm{pos}\big(c(t_1, t_2)\big) = \big\{\varepsilon\big\} \cup \big\{1 \cdot w \mid w \in \mathrm{pos}(t_1)\big\} \cup \big\{2 \cdot w \mid w \in \mathrm{pos}(t_2)\big\} \enspace .$$

We let $\mathrm{leaves}(t) = \{w \in \mathrm{pos}(t) \mid w \cdot 1 \notin \mathrm{pos}(t)\}$ be the set of leaf positions in t, and $\mathrm{ht}(t) = \max_{w \in \mathrm{leaves}(t)} |w|$ be the height of the tree t. The subtree of t at position $w \in \mathrm{pos}(t)$ is denoted by $t|_w$, and the label of t at position w is denoted by $t(w)$. Moreover, $t[t']_w$ denotes the tree obtained from t by replacing the subtree at position w by the tree $t' \in T_{\Sigma_2,\Sigma_1}(\Sigma_0)$. Given $\Delta \subseteq \Sigma_2 \cup \Sigma_1 \cup \Sigma_0$, let $\mathrm{pos}_\Delta(t) = \{w \in \mathrm{pos}(t) \mid t(w) \in \Delta\}$. We simply write $\mathrm{pos}_\delta(t)$ instead of $\mathrm{pos}_{\{\delta\}}(t)$.

We reserve the use of the symbol $\square$. The set $C_{\Sigma_2,\Sigma_1}(\Sigma_0)$ of *contexts* contains all trees of $T_{\Sigma_2,\Sigma_1}(\Sigma_0 \cup \{\square\})$, in which the special symbol $\square$ occurs exactly once. Let $C \in C_{\Sigma_2,\Sigma_1}(\Sigma_0)$. Since $\mathrm{pos}_\square(C)$ contains one element, we often identify it with its only element. To save space, we write tC for $C[t]_w$, where $w = \mathrm{pos}_\square(C)$.[2]

A *relabeling* is a mapping $\rho\colon (\Sigma_2 \cup \Sigma_1 \cup \Sigma_0) \to \mathcal{P}_+(\Delta)$ for some alphabet Δ.[3] It induces a mapping $\widehat{\rho}\colon T_{\Sigma_2,\Sigma_1}(\Sigma_0) \to \mathcal{P}_+(T_{\Delta,\Delta}(\Delta))$ for every $t \in T_{\Sigma_2,\Sigma_1}(\Sigma_0)$ by

$$\widehat{\rho}(t) = \big\{u \in T_{\Delta,\Delta}(\Delta) \mid \mathrm{pos}(u) = \mathrm{pos}(t),\, \forall w \in \mathrm{pos}(u)\colon u(w) \in \rho\big(t(w)\big)\big\} \enspace .$$

In the following, we again do not distinguish between the relabeling ρ and its induced mapping $\widehat{\rho}$ on trees. A *simple (monadic) context-free tree grammar* [19, 20] (sCFTG) is a system $G = (N, \Sigma, I, P)$ such that (i) $N = N_1 \cup N_0$, where N_1 and N_0 are alphabets of unary and nullary *nonterminals*, respectively, (ii) $\Sigma = \Sigma_2 \cup \Sigma_0$, where Σ_2 and Σ_0 are alphabets of internal and leaf *terminal symbols*, respectively, such that $N \cap \Sigma = \emptyset$, (iii) $I \subseteq N_0$ are nullary *start nonterminals*, and (iv) P is a finite set of *productions* such that

$$P \subseteq \big(N_0 \times T_{\Sigma_2,N_1}(\Sigma_0 \cup N_0)\big) \cup \big(N_1 \times C_{\Sigma_2,N_1}(\Sigma_0 \cup N_0)\big) \enspace .$$

The grammar is called monadic, because there are only nullary and unary nonterminals; simple means that the rules are linear and nondeleting, so all subtrees of a nonterminal on the left side of a rule have to appear exactly once on the right side. If $N_1 = \emptyset$, then G is a *regular tree grammar* (RTG). We write productions (n, r) as $n \to r$. Next, we define the rewrite semantics [2] for the sCFTG G. For arbitrary $\xi, \zeta \in T_{\Sigma_2,N_1}(\Sigma_0 \cup N_0)$ and positions $w \in \mathrm{pos}(\xi)$ we let $\xi \Rightarrow_{G,w} \zeta$ if there exists a production $n \to r \in P$ such that

- $\xi|_w = n$ and $\zeta = \xi[r]_w$ with $n \in N_0$, or
- $\xi|_w = n(\xi')$ and $\zeta = \xi[\xi' r]_w$ with $n \in N_1$ and $\xi' \in T_{\Sigma_2,N_1}(\Sigma_0 \cup N_0)$.

We write $\xi \Rightarrow_G \zeta$ if there exists $w \in \mathrm{pos}(\xi)$ such that $\xi \Rightarrow_{G,w} \zeta$. The tree language $\mathcal{F}(G)$ generated by G is $\mathcal{F}(G) = \{t \in T_{\Sigma_2,\emptyset}(\Sigma_0) \mid \exists n_0 \in I\colon n_0 \Rightarrow_G^+ t\}$, where $\Rightarrow_G^+$ is the transitive closure of $\Rightarrow_G$. The tree languages generated by sCFTGs are *context-free*,[4] and a tree language $\mathcal{F}$ is *regular* if and only if there exists an RTG G such that $\mathcal{F} = \mathcal{F}(G)$. A detailed introduction to trees and tree languages can be found in [10].

[2] This order tC is beneficial for arguments C (see Section 3).

[3] We require that each input symbol can be relabeled.

[4] Note that this is not an equivalence. There are context-free tree languages that are not generated by any sCFTG.

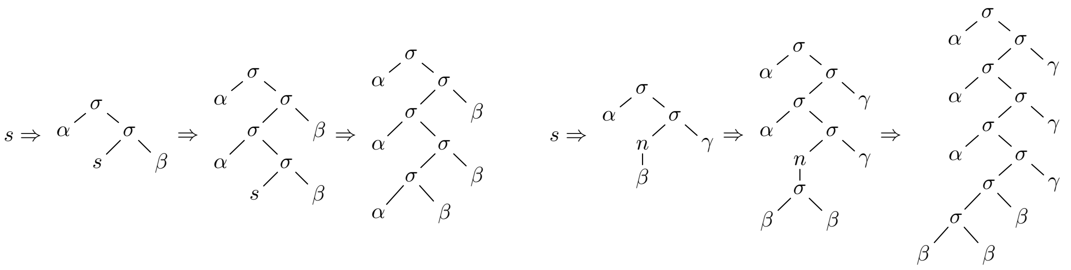


Figure 1 Derivations using the RTG G_1 (left) and the sCFTG G_2 (right) of Examples 1 and 2, respectively.

▶ **Example 1.** The regular tree grammar $G_1 = (N, \Sigma, I, P)$ with $N = N_0 = I = \{s\}$, $\Sigma_2 = \{\sigma\}$, $\Sigma_0 = \{\alpha, \beta\}$, and $P = \{s \to \sigma(\alpha, \sigma(s, \beta)),\ s \to \sigma(\alpha, \beta)\}$ generates the leaf language $\{\alpha^n \beta^n \mid n \geq 1\}$. Note that because it is an RTG, all nonterminals are nullary and thus leaves, which is similar to the property of right-linearity that can be encountered in CFGs.

Two important facts concerning the regular tree languages are that they properly include the derivation tree languages of CFGs and that their leaf languages are exactly the context-free languages.

▶ **Example 2.** The sCFTG $G_2 = (N, \Sigma, I, P)$ with $N = N_0 = I = \{s\}$, $N_2 = \{n\}$, $\Sigma_2 = \{\sigma\}$, $\Sigma_0 = \{\alpha, \beta, \gamma\}$ and

$$\begin{aligned} P = \big\{ & s \to \sigma(\alpha, \sigma(\beta, \gamma)),\ s \to \sigma(\alpha, \sigma(n(\beta), \gamma)), \\ & n \to \sigma(\alpha, \sigma(n(\sigma(\square, \beta)), \gamma)),\ n \to \sigma(\alpha, \sigma(\sigma(\square, \beta), \gamma))\big\} \end{aligned}$$

generates the leaf language $\{\alpha^n \beta^n \gamma^n \mid n \geq 1\}$. Since G_2 is simple, the placeholder $\square$, which indicates the new position of the subtree under the unary nonterminal symbol n, appears exactly once on the right side of the respective rules.

3 Combinatory Categorial Grammars

Combinatory categorial grammars (CCGs) extend the classical categorial grammars of Ajdukiewicz and Bar-Hillel [4] by rules inspired by combinatory logic [8]. Here, as in most of the formal work on CCGs, we restrict our attention to the rules of composition, which are based on the **B**-combinator.

Let A be an alphabet, and let $\mathcal{C}(A) = T_{S,\emptyset}(A)$, where $S = \{/, \backslash\}$ is the set of slashes. The elements of $\mathcal{C}(A)$ are called *categories* (over A), of which the elements of $A \subseteq \mathcal{C}(A)$ are *atomic*. We write categories using infix notation, omitting unnecessary parentheses based on the convention that slashes are left-associative. Thus every category takes the form $c = a|_1 c_1 \cdots |_k c_k$ where $a \in A$, $|_i \in S$, and $c_i \in \mathcal{C}(A)$, for all $i \in [k]$. The atomic category a is called the *target* of c and the slash–argument pairs $|_i c_i$ are called the *arguments* of c. If $c_i \in A$ for all $i \in [k]$, we call c a *first-order category*. The set of all first-order categories over A is denoted by $\mathcal{C}_f(A)$. The number k is called the *arity* of c. Note that, from the tree perspective, the sequence of arguments is a context $\alpha = \square|_1 c_1 \cdots |_k c_k$. The number k is the *length* of α; we write it as $|\alpha|$. We let $\mathcal{A}(A) \subseteq C_{S,\emptyset}(A)$ be the set of all argument contexts (over A). Finally, for every $k \in \mathbb{N}$, we let $\mathcal{C}(A, k) = \{\, c \in \mathcal{C}(A) \mid \text{arity}(c) \leq k \,\}$ and $\mathcal{A}(A, k) = \{\, \alpha \in \mathcal{A}(A) \mid |\alpha| \leq k \,\}$.

Intuitively, a category c/c' can be combined with a category c' to its right to become c; similarly, a category $c\backslash c'$ can be combined with c' to its left. Formally, given an alphabet A and $k \in \mathbb{N}$, a *rule of degree k over A* takes one of two possible forms [28]:

$$ax/c,\ c|_1c_1 \cdots |_kc_k \to ax|_1c_1 \cdots |_kc_k \qquad \text{(forward rule)}$$
$$c|_1c_1 \cdots |_kc_k,\ ax\backslash c \to ax|_1c_1 \cdots |_kc_k \qquad \text{(backward rule)}$$

where $a \in A$, $c \in \mathcal{C}(A) \cup \{y\}$, and $|_i \in S$ and $c_i \in \mathcal{C}(A) \cup \{y_i\}$ for every $i \in [k]$. The category $ax|c$ with $| \in \{/, \backslash\}$ is called the *primary input category* and the other category $c|_1c_1 \cdots |_kc_k$ is the *secondary input category* of the rule. The categories $c, c_1, \ldots, c_k$ can thus be either concrete categories from $\mathcal{C}(A)$ or a category variable $\{y, y_1, \ldots, y_k\}$ that will match each category from $\mathcal{C}(A)$. Similarly, the argument context variable x will match each argument context of $\mathcal{A}(A)$. We let $\mathcal{R}(A)$ be the set of all rules over A, and for every $k \in \mathbb{N}$ let $\mathcal{R}(A, k)$ be the finite set of all generic (i.e. always using variables instead of concrete categories) rules over A with degree at most k. Rules of degree 0 are called *application rules*, whereas rules of higher degree are called *composition rules*. A *rule system* is a pair $\Pi = (A, R)$ consisting of an alphabet A and a finite set $R \subseteq \mathcal{R}(A)$ of rules over A. A *ground instance* of a rule r is obtained by substituting concrete categories for the variables $\{y, y_1, \ldots\}$ and a concrete argument context for the variable x in r. The set of all ground instances of Π induces a relation $\to_\Pi \subseteq \mathcal{C}(A)^2 \times \mathcal{C}(A)$, which extends to a relation $\Rightarrow_\Pi \subseteq \mathcal{C}(A)^* \times \mathcal{C}(A)^*$ by $\Rightarrow_\Pi = \{\, (\varphi\, c\, c'\, \psi,\ \varphi\, c''\, \psi) \mid \varphi, \psi \in \mathcal{C}(A)^*;\ c, c' \to_\Pi c'' \,\}$.

▶ **Example 3.** Consider the rule $r = Dx/D,\ D/E\backslash C \to Dx/E\backslash C$, where $\{C, D, E\}$ are atoms and x is an argument context variable. A possible ground instance of this rule is $D/C/E/D,\ D/E\backslash C \to D/C/E/E\backslash C$, where x was replaced by the argument context $\square/C/E$. The primary input category $c_1 = D/C/E/D$ has target D and arguments $/C$, $/E$, and $\backslash D$. As c_1 takes three atomic categories as arguments, it is a first-order category and $\mathit{arity}(c_1) = 3$. The rule degree is determined by the number of arguments replacing the last argument of the primary input category, so r has degree $k = 2$. Note that $D/C/E/D$ is short for $((D/C)/E)/D$, which is different from $(D/C)/(E/D)$. The rules $r' = Dx/(E/D),\ E/D\backslash C \to Dx\backslash C$ and $r'' = Dx/(E/D),\ E/D\backslash(C/C) \to Dx\backslash(C/C)$ both have first degree.

▶ **Definition 4** ([28]). *A* combinatory categorial grammar *(CCG) is a tuple $G = (\Sigma, A, R, I, L)$ consisting of an alphabet Σ of* input symbols*, a* rule system *(A, R), a set $I \subseteq A$ of* initial categories*, and a finite relation $L \subseteq \Sigma \times \mathcal{C}(A)$ called* lexicon*. It is a k-*CCG *[resp.* pure k-CCG*], for $k \in \mathbb{N}$, if each $r \in R$ has degree at most k [resp. if $R = \mathcal{R}(A, k)$].*

▶ **Example 5.** The classical categorial grammars of Ajdukiewicz and Bar-Hillel [4], which are also called *AB-grammars*, are 0-CCGs. However, the term 0-CCG is more general since as opposed to AB-grammars, they allow rule restrictions (i.e. they are not necessarily pure). As a concrete example, let $G_3 = (\Sigma, A, \mathcal{R}(A, 0), I, L)$ be the CCG given by the input alphabet $\Sigma = \{c, d\}$, the atomic categories $A = \{C, D\}$, the set of initial categories $I = \{C\}$, and the lexicon L with $L(c) = \{C/D,\ C/D/C\}$ and $L(d) = \{D\}$. Clearly, it is a 0-CCG. For a slightly more involved example containing rule restrictions, see Example 15.

▶ **Definition 6.** *A combinatory categorial grammar $G = (\Sigma, A, R, I, L)$* accepts *the category sequences $\mathcal{C}(G) \subseteq \mathcal{C}(A)^*$ and the string language $\mathcal{L}(G) \subseteq \Sigma^*$, where*

$$\mathcal{C}(G) = \{\varphi \in \mathcal{C}(A)^* \mid \exists a_0 \in I \colon \varphi \Rightarrow^*_{(A,R)} a_0\} \qquad \textit{and} \qquad \mathcal{L}(G) = L^{-1}(\mathcal{C}(G)) \ .$$

A tree $t \in T_{\mathcal{C}(A),\emptyset}(L(\Sigma))$ is a derivation tree *of G if $t(w \cdot 1),\ t(w \cdot 2) \to_{(A,R)} t(w)$ for every $w \in \mathrm{pos}(t) \setminus \mathrm{leaves}(t)$. The set of all such trees is denoted by $\mathcal{D}(G)$.*

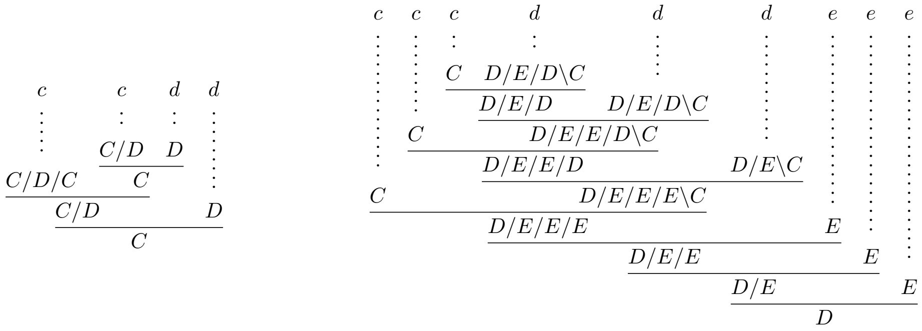


Figure 2 Derivations using the AB-grammar G_3 (left) and the CCG G_4 (right) of Examples 5 and 15, respectively.

The grammar of Example 5 accepts $\mathcal{L}(G_3) = \{c^i d^i \mid i \geq 1\}$, which is context-free but not regular. A derivation tree for the string *ccdd* is shown in Figure 2. We draw derivation trees according to the standard conventions for CCGs, so the root is drawn at the bottom. The dotted lines visualize the input symbol–category mapping implemented by the lexicon. Overall, G_3 accepts the category sequences $\mathcal{C}(G_3) = \{(C/D/C)^{i-1} \cdot (C/D) \cdot D^i \mid i \geq 1\}$.

The language accepted by a CCG is obtained by relabeling the leaf categories of the derivation trees using the lexicon. For the accepted tree language we similarly allow a relabeling to avoid the restriction to the particular symbols of $\mathcal{C}(A)$.

▶ **Definition 7.** *Let $G = (\Sigma, A, R, I, L)$ be a CCG and $\rho\colon \mathcal{C}(A) \to \mathcal{P}_+(\Delta)$ be a relabeling. They* accept *the tree language $\mathcal{F}_\rho(G) = \{\rho(d) \in T_{\Delta,\emptyset}(\Delta) \mid d \in \mathcal{D}(G),\, d(\varepsilon) \in I\}$. A tree language $\mathcal{F} \subseteq T_{\Delta,\emptyset}(\Delta)$ is* acceptable *by the CCG G if there exists a relabeling $\rho'\colon \mathcal{C}(A) \to \mathcal{P}_+(\Delta)$ such that $\mathcal{F} = \mathcal{F}_{\rho'}(G)$.*

Because $L(\Sigma)$ is finite, there exists $k \in \mathbb{N}$ such that $L(\Sigma) \subseteq \mathcal{C}(A, k)$. The least such integer k is called the *arity of* L and denoted by $\operatorname{arity}(L)$; i.e. $\operatorname{arity}(L) = \max\{\operatorname{arity}(c) \mid c \in L(\Sigma)\}$. If $L = \emptyset$, then we let $\operatorname{arity}(L) = 0$.

4 0-CCGs

Let $G = (\Sigma, A, R, I, L)$ be a 0-CCG. An important property of 0-CCGs is that each category that occurs in a derivation tree has arity at most $\operatorname{arity}(L)$. Thus, derivation trees are built over a finite set of symbols.

▶ **Theorem 8** (see [4] and [25, Proposition 3.25])**.** *The string languages accepted by 0-CCGs are exactly the ε-free context-free languages. Moreover, for each 0-CCG G the derivation tree language $\mathcal{D}(G)$ and the accepted tree languages are regular.*

To characterize the tree languages accepted by 0-CCGs, we need to introduce an additional structural property of the derivation tree language $\mathcal{D}(G)$ and the acceptable tree languages. Roughly speaking, the *min-height* $\operatorname{mht}(t)$ of a tree t is the minimal length of a path from the root to a leaf. Recall that the height coincides with the maximal length of those paths. For all alphabets Σ_2 and Σ_0, let $\operatorname{mht}\colon T_{\Sigma_2,\emptyset}(\Sigma_0) \to \mathbb{N}$ be such that $\operatorname{mht}(a) = 0$ and $\operatorname{mht}(c(t_1, t_2)) = 1 + \min(\operatorname{mht}(t_1), \operatorname{mht}(t_2))$ for all $a \in \Sigma_0$, $c \in \Sigma_2$, and $t_1, t_2 \in T_{\Sigma_2,\emptyset}(\Sigma_0)$. A tree $t \in T_{\Sigma_2,\emptyset}(\Sigma_0)$ is *universally* mht*-bounded by* $h \in \mathbb{N}$ if $\operatorname{mht}(t|_w) \leq h$ for every $w \in \operatorname{pos}(t)$. Finally, a tree language $\mathcal{F} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$ is *universally* mht*-bounded by* h if every $t \in \mathcal{F}$ is

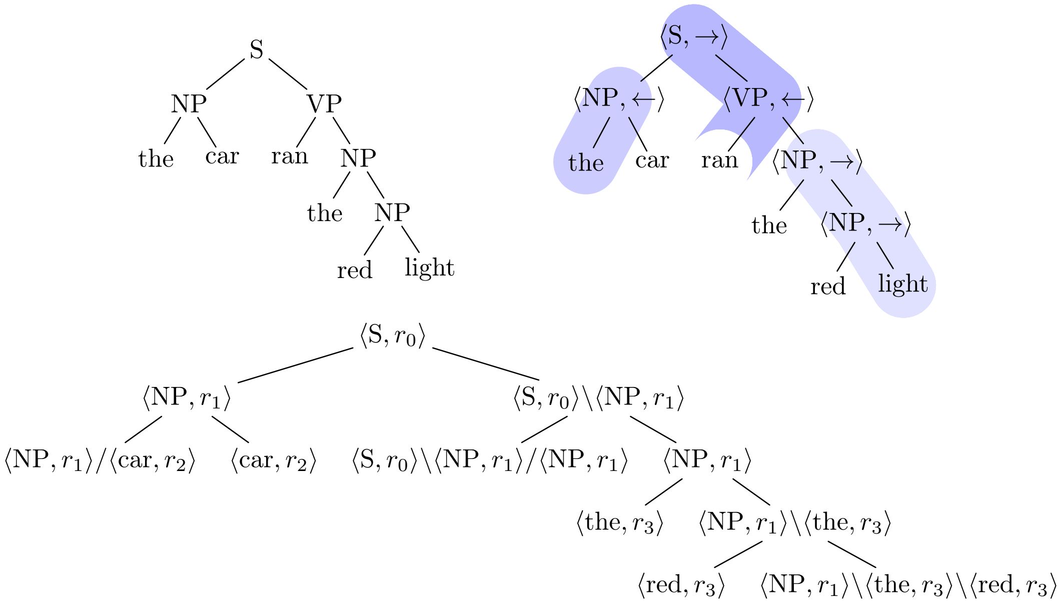


Figure 3 Decomposition into spinal runs and the corresponding derivation tree of the 0-CCG.

universally mht-bounded by h, and it is *universally* mht-*bounded* if there exists $h \in \mathbb{N}$ such that it is universally mht-bounded by h. Note that "universally mht-bounded" is a purely structural property of a tree as it only depends on the shape of the tree and is completely agnostic about the node labels. It is thus preserved by the application of a relabeling. Consequently, $\rho(\mathcal{F})$ is universally mht-bounded by h if and only if $\mathcal{F}$ is universally mht-bounded by h for every tree language $\mathcal{F} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$ and relabeling $\rho \colon (\Sigma_2 \cup \Sigma_0) \to \mathcal{P}_+(\Delta)$.

Let us reconsider the 0-CCG G_3 of Example 5. The set $\mathcal{D}(G_3)$ is universally mht-bounded by 1 (see Figure 2). It turns out that exactly the universally mht-bounded regular tree languages are acceptable by 0-CCGs. We already observed that the tree languages acceptable by 0-CCGs are regular, but for the converse we have to exploit the universal mht-bound. We utilize those short paths to a leaf to decompose the tree into spines, which are short paths in the tree that lead from a node to a leaf and are never longer than the universal min-height. The primary categories for the applications are placed along those spines and each spine terminates in an atomic category that can be combined with the category from another spine. The idea of the construction is illustrated in Figure 3. This close relation and the good closure properties of regular tree languages allow us to derive a number of closure results for the tree languages acceptable by 0-CCGs (see Table 1).

▶ **Theorem 9.** *Let $\mathcal{F} \subseteq T_{\Sigma_2,\emptyset}(\Sigma_0)$ be a tree language. Then $\mathcal{F}$ is acceptable by some 0-CCG if and only if it is regular and universally* mht-*bounded.*

We have seen that, while basic categorial grammars and context-free grammars are weakly equivalent, they are not strongly equivalent when considered as tree-generating devices. More specifically, the class of derivation tree languages of basic categorial grammars are a proper subclass of the class of local tree languages (i.e. derivation tree languages of context-free grammars). This result is similar to a result by Schabes et al. [21] showing that context-free grammars are not closed under strong lexicalization, meaning that there are context-free grammars such that no lexicalized grammar[5] generates the same derivation tree language.

[5] A CFG is called lexicalized if every production contains a terminal symbol.

Table 1 Closure properties of the tree languages acceptable by 0-CCGs and 1-CCGs.

closure \ class	regular tree languages = tree languages acceptable by 1-CCGs	tree languages acceptable by 0-CCGs
union	✓	✓
intersection	✓	✓
complement	✓	✗
relabeling	✓	✓
α-concatenation [10]	✓	✓
α-iteration [10]	✓	✗

5 1-CCGs

In this section, we will consider 1-CCGs, which allow rules of degree at most 1. Thus, the secondary input categories appearing in their derivation tree languages have at most one additional argument after the category consumed by the composition. We will show that the 1-CCGs accept exactly the regular tree languages by showing inclusion in both directions.

▶ **Lemma 10.** *For each 1-CCG G the derivations $\mathcal{D}(G)$ and the accepted tree language are regular.*

The following lemma establishes a normal form for regular tree grammars that is easily achieved using standard techniques. For every $m \in \mathbb{N}$, let $\mathbb{Z}_m = \{i \in \mathbb{N} \mid 0 \leq i < m\}$.

▶ **Lemma 11.** *For each RTG G there exists an equivalent RTG $G' = (\mathbb{Z}_m, \Sigma, I', P')$ in normal form, in which every nonterminal $n \in \mathbb{Z}_m$ generates a uniquely defined terminal symbol σ_n; i.e. for all $n \in \mathbb{Z}_m$ there exists $\sigma_n \in \Sigma$ such that $t(\varepsilon) = \sigma_n$ for all $t \in T_\Sigma$ with $n \Rightarrow^+_{G'} t$.*

Given an RTG $G = (\mathbb{Z}_m, \Sigma, I, P)$ in the normal form of Lemma 11, we are allowed to regard only the nonterminals of G when constructing an equivalent 1-CCG. Our goal is to find a 1-CCG $G' = (\Sigma', A, R, I', L)$ and a projection $\pi\colon \mathcal{C}(A) \to \mathbb{Z}_m$ such that $\mathcal{F}(G) = \mathcal{F}_{\pi' \circ \pi}(G')$. Because π maps from categories to nonterminals, but $\mathcal{F}(G)$ is labeled by terminal symbols, we need the projection $\pi'\colon \mathbb{Z}_m \to \Sigma$ to map from nonterminals to terminals. This function is well-defined due to the constraint on G, that each nonterminal generates a single terminal.

Given a production $n \to \sigma(n_1, n_2) \in P$ and a projection $\pi\colon \mathcal{C}(A) \to \mathbb{Z}_m$, we need categories $c_1 \in \pi^{-1}(n_1)$ and $c_2 \in \pi^{-1}(n_2)$ for each category $c \in \pi^{-1}(n)$ such that $c_1, c_2 \to c$ is a valid ground instance of a rule in R. This ensures that each category can be derived by the composition of two categories mapped to matching nonterminals. We only regard first-order categories with at most one argument due to the restriction on 1-CCGs. Starting from any nonterminal, the productions P allow the derivation of at most all ordered pairs of nonterminals. The number of ordered pairs $\mathbb{Z}^2_m$ increases quadratically in m, whereas the number of different composition input pairs resulting in a fixed category increases only linearly in $|A|$. The *category matrix* depicted in Figure 4 illustrates that a first-order category with one argument is the result of the forward compositions of $|A|$ different category pairs. In addition to composition rules, application rules are neccessary to obtain an atomic initial category. Based on these observations, we construct a 1-CCG G' with m^2 atoms in the following way.

▶ **Definition 12.** *Given an RTG $G = (\mathbb{Z}_m, \Sigma, I, P)$ in the normal form of Lemma 11, we construct the 1-CCG $G' = (\{x\}, \mathbb{Z}_m^2, R, \pi^{-1}(I) \cap \mathbb{Z}_m^2, L)$ with*

$$\begin{aligned} R &= \big\{a/b,\, b \to a \mid \pi(a) \to \sigma\big(\pi(a/b), \pi(b)\big) \in P,\, \sigma \in \Sigma_2,\, a, b \in \mathbb{Z}_m^2\big\} \\ &\quad \cup \big\{a/b,\, b/c \to a/c \mid \pi(a/c) \to \sigma\big(\pi(a/b), \pi(b/c)\big) \in P,\, \sigma \in \Sigma_2,\, a, b, c \in \mathbb{Z}_m^2\big\} \\ L &= \{(x, a) \mid a \in \mathcal{C}(A, 1) \cap \mathcal{C}_f(A),\, \pi(a) \to \alpha \in P,\, \alpha \in \Sigma_0,\, a \in \mathbb{Z}_m^2\} \end{aligned}$$

where the mapping $\pi\colon (\mathbb{Z}_m^2 \cup \{n/n' \mid n, n' \in \mathbb{Z}_m^2\}) \to \mathbb{Z}_m$ is given by $\pi\big((i,j)\big) = i$ and $\pi\big((i,j)/(i',j')\big) = i + j' \bmod m$ for all $i, i', j, j' \in \mathbb{Z}_m$.

▶ **Lemma 13.** *For each RTG G there exists a 1-CCG G' accepting the tree language $\mathcal{F}(G)$.*

Proof. We have to establish that the 1-CCG $G' = (\{x\}, \mathbb{Z}_m^2, R, \pi^{-1}(I), L)$ of Definition 12 accepts the tree language $\mathcal{F}(G)$ of RTG $G = (\mathbb{Z}_m, \Sigma, I, P)$ using the relabeling $\pi' \circ \pi$. The category $c = (i,j)/(i',j')$ is the result of the forward composition of $(i,j)/(k,l)$ and $(k,l)/(i',j')$, where $i, i', j, j', k, l \in \mathbb{Z}_m$. Figure 4 illustrates the projection π by means of a *projection matrix*, which is a category matrix with categories replaced according to the projection. Row and column labels follow lexicographic order. When we slice it evenly into blocks of size $m \times m$, we can observe that the entries in the rows cycle through the nonterminals, whereas in a single column, each block has only a single nonterminal in all m entries. This is because the value of j' changes in every entry, whereas the value of i changes only every m entries. Nonetheless, a complete column of the whole projection matrix contains all m nonterminals. Relabeling in this manner ensures that all pairs of nonterminals are covered by arbitrary pairs of row and column. These are determined by the result category.

Given a category $(i,j)/(i',j')$ and an ordered pair (g,h) of nonterminals, we need to verify that there exist $k, l \in \mathbb{Z}_m$ with $\pi((i,j)/(k,l)) = g$ and $\pi((k,l)/(i',j')) = h$. Since $g = (i{+}l) \bmod m$ and $h = (k{+}j') \bmod m$, we obtain $l = (g{-}i) \bmod m$ and $k = (h{-}j') \bmod m$. Furthermore, given an arbitrary atom (i,j) and nonterminals $g, h \in \mathbb{Z}_m$, we want to find a category $(i,j)/(k,l)$ and an atom (k,l) such that $\pi((i,j)/(k,l)) = g$ and $\pi((k,l)) = h$. From the definition of the projection, $\pi((k,l)) = k$, so we have $k = h$ and $l = (g - i) \bmod m$.

We relabel $\mathcal{F}(G')$ using $\pi' \circ \pi$ as described above. Due to the fact that the categories occurring in derivation trees of G' cannot have higher order or arity greater than 1, they never leave the domain of π. As a result, we were able to construct a 1-CCG accepting the tree language $\mathcal{F}(G)$. Thus, for each regular tree language, we can construct a 1-CCG accepting it. ◀

▶ **Theorem 14.** *The tree languages accepted by 1-CCGs are exactly the regular tree languages.*

6 Inclusion in the Context-Free Tree Languages

In this section, we want to relate the derivation tree languages of CCGs to the context-free tree languages. However, this is complicated by the presence of potentially infinitely many categories. Let us illustrate the problem first.

▶ **Example 15.** Let $G_4 = (\Sigma, A, R, \{D\}, L)$ be the 3-CCG given by the alphabet $\Sigma = \{c, d, e\}$, the atomic categories $A = \{C, D, E\}$, the lexicon L with $L(c) = \{C\}$, $L(d) = \{D/E\backslash C, D/E/D\backslash C\}$, $L(e) = \{E\}$, and the rule set R consisting of the rules

$$\begin{aligned} &Dx/D,\, D/E/D\backslash C \to Dx/E/D\backslash C && Dx/E,\, E \to Dx \\ &Dx/D,\, D/E\backslash C \to Dx/E\backslash C && C,\, Dx\backslash C \to Dx \end{aligned}$$

	a_0	a_1	a_2	a_3
a_0	a_0/a_0	a_0/a_1	a_0/a_2	a_0/a_3
a_1	a_1/a_0	a_1/a_1	a_1/a_2	a_1/a_3
a_2	a_2/a_0	a_2/a_1	a_2/a_2	a_2/a_3
a_3	a_3/a_0	a_3/a_1	a_3/a_2	a_3/a_3

	$(0,0)$	$(0,1)$	$(0,2)$	$(1,0)$	$(1,1)$	$(1,2)$	$(2,0)$	$(2,1)$	$(2,2)$
$(0,0)$	0	1	2	0	1	2	0	1	2
$(0,1)$	0	1	2	0	1	2	0	1	2
$(0,2)$	0	1	2	0	1	2	0	1	2
$(1,0)$	1	2	0	1	2	0	1	2	0
$(1,1)$	1	2	0	1	2	0	1	2	0
$(1,2)$	1	2	0	1	2	0	1	2	0
$(2,0)$	2	0	1	2	0	1	2	0	1
$(2,1)$	2	0	1	2	0	1	2	0	1
$(2,2)$	2	0	1	2	0	1	2	0	1

Figure 4 The category matrix (left) contains all first-order categories of arity 1 with only forward slashes in a CCG with four atoms. Each category is the result of the forward composition of a category taken from the same row and one from the same column, respectively. The i-th entry of each row can be combined with the i-th entry of each column. Thus, each category is the result of four different forward compositions. The projection matrix (right) shows a 1-CCG with nine atomic categories after relabeling using projection $\pi : \mathcal{C}(A) \to \mathbb{Z}_3$, obtained from an RTG with three nonterminals by applying Definition 12. Suppose we want to find two categories projected to nonterminals $(g,h) = (0,2)$ whose composition yields $(i,j)/(i',j') = (0,1)/(0,1)$. These are categories $(0,1)/(1,0)$ and $(1,0)/(0,1)$ since $(k,l) = ((h-j') \bmod 3, (g-i) \bmod 3) = ((2-1) \bmod 3, (0-0) \bmod 3) = (1,0)$.

where $x \in \mathcal{A}(A)$. From a few sample derivation trees (see Figure 2) we can convince ourselves that G_4 accepts the string language $\mathcal{L}(G_4) = \{\, c^i d^i e^i \mid i \geq 1 \,\}$, which shows that 3-CCGs can accept non-context-free string languages. In addition, the derivation trees $\mathcal{D}(G_4)$ contain infinitely many categories as labels.

Since the classical tree language theory only handles finitely many labels, we switch to a different representation and consider *rule trees*. To simplify the notation, we introduce the following shorthands. We let $\mathrm{T} = T_{R,\emptyset}\big(L(\Sigma)\big)$ be the set of all potential rule trees (see Definition 16), and for all alphabets N_1 and N_0 we let $\mathrm{SF}(N_1, N_0) = T_{R,N_1}\big(L(\Sigma) \cup N_0\big)$ be the forms of a sCFTG with unary nonterminals N_1 and nullary nonterminals N_0.

▶ **Definition 16.** *Let $G = (\Sigma, A, R, I, L)$ be a CCG. A tree $t \in \mathrm{T}$ is a* rule tree *of G if $\mathrm{cat}_G(t) \in I$, where $\mathrm{cat}_G : \mathrm{T} \to \mathcal{C}(A)$ is the partial function that is inductively defined by*

- *$\mathrm{cat}_G(c) = c$ for all $c \in L(\Sigma)$,*
- *$\mathrm{cat}_G\big((ax/c,\ c\gamma \to ax\gamma)(t_1, t_2)\big) = a\alpha\gamma$ for all trees $t_1, t_2 \in \mathrm{T}$ such that $\mathrm{cat}_G(t_1) = a\alpha/c$ and $\mathrm{cat}_G(t_2) = c\gamma$, and*
- *$\mathrm{cat}_G\big((c\gamma,\ ax\backslash c \to ax\gamma)(t_1, t_2)\big) = a\alpha\gamma$ for all trees $t_1, t_2 \in \mathrm{T}$ such that $\mathrm{cat}_G(t_1) = c\gamma$ and $\mathrm{cat}_G(t_2) = a\alpha\backslash c$.*

The set of all rule trees of G is denoted by $\mathcal{R}(G)$.

The rule trees provide a natural encoding of the (successful) derivation trees of a CCG using only finitely many labels. More precisely, there is an (obvious) bijection between the derivation trees $\mathcal{D}(G)$ and the domain of the function cat_G.

In the following, let $G = (\Sigma, A, R, I, L)$ be a CCG. Our goal is to construct an sCFTG that generates exactly the rule tree language $\mathcal{R}(G)$. To this end, we first need to limit the arity of the categories. Let $k \in \mathbb{N}$ be the maximal arity of a category in

$$I \cup L(\Sigma) \cup \{c\gamma \mid ax/c,\ c\gamma \to ax\gamma \in R\} \cup \{c\gamma \mid c\gamma,\ ax\backslash c \to ax\gamma \in R\}\,,$$

i.e. the maximal arity of the categories that occur in the lexicon, as initial category, or as the secondary premise of a rule of R. Roughly speaking, the constructed sCFTG will use the categories $\mathcal{C}(A, k)$ as nullary nonterminals and tuples $\langle a, |c, \gamma\rangle$ consisting of an

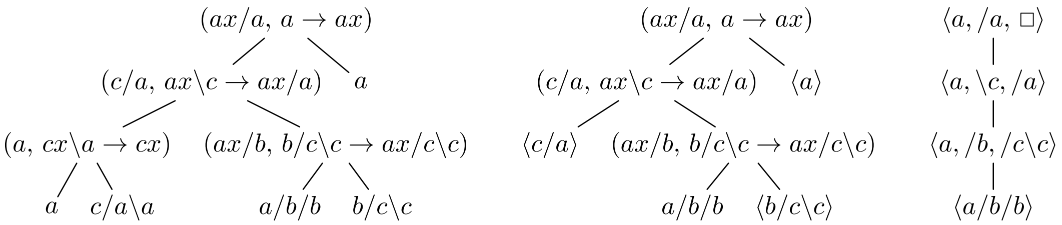

$(ax/a,\ a \to ax)$
$(c/a,\ ax\backslash c \to ax/a)$
a
$(a,\ cx\backslash a \to cx)$
$(ax/b,\ b/c\backslash c \to ax/c\backslash c)$
a
$c/a\backslash a$
$a/b/b$
$b/c\backslash c$
$(ax/a,\ a \to ax)$
$(c/a,\ ax\backslash c \to ax/a)$
$\langle a\rangle$
$\langle c/a\rangle$
$(ax/b,\ b/c\backslash c \to ax/c\backslash c)$
$a/b/b$
$\langle b/c\backslash c\rangle$
$\langle a, /a, \Box\rangle$
$\langle a, \backslash c, /a\rangle$
$\langle a, /b, /c\backslash c\rangle$
$\langle a/b/b\rangle$

Figure 5 Rule tree t (without lexical entries), $\mathrm{spinal}(t)$, and its encoding $\mathrm{enc}(t)$.

atomic category $a \in A$, a single argument $|c$ with $| \in S$ and $c \in \mathcal{C}(A,k)$, and an argument tree $\gamma \in \mathcal{A}(A,k)$ as unary nonterminals. Recall that we write substitutions $\alpha[t]$ as $t\alpha$ for $\alpha \in \mathcal{A}(A)$ and $t \in \mathcal{C}(A) \cup \mathcal{A}(A)$.

▶ **Definition 17.** *We construct the sCFTG $G' = (N_1 \cup N_0, R \cup L(\Sigma), I', P)$ with*

- $N_1 = \{\langle a, |c, \gamma\rangle \mid a \in A,\ | \in S,\ c \in \mathcal{C}(A,k),\ \gamma \in \mathcal{A}(A,k)\}$ *and* $N_0 = \{\langle c\rangle \mid c \in \mathcal{C}(A,k)\}$*,*
- $I' = \{\langle a_0\rangle \mid a_0 \in I\}$*, and*
- *the following set P of productions*

$$\begin{aligned}
P = \big\{\langle c\rangle \to c \mid c \in L(\Sigma)\big\} \cup & \qquad (1)\\
\big\{\langle a, /c, \gamma\rangle \to s\big(\Box, \langle c\gamma\rangle\big) \mid s = \big(ax/c,\ c\gamma \to ax\gamma\big) \in R\big\} \cup & \qquad (2)\\
\big\{\langle a, \backslash c, \gamma\rangle \to s\big(\langle c\gamma\rangle, \Box\big) \mid s = \big(c\gamma,\ ax\backslash c \to ax\gamma\big) \in R\big\} \cup & \qquad (3)\\
\big\{\langle a\alpha\gamma\rangle \to \langle a, |c, \gamma\rangle\big(\langle a\alpha|c\rangle\big) \mid a \in A,\ \alpha, \gamma \in \mathcal{A}(A),\ | \in S,\ c \in \mathcal{C}(A,k), & \\
|\alpha| < k,\ |\alpha\gamma| \le k\big\} \cup & \qquad (4)\\
\big\{\langle a, |c, \gamma\rangle \to \langle a, |'c', \Box\rangle\big(\langle a, |c, \gamma|'c'\rangle(\Box)\big) & \\
\mid a \in A,\ |, |' \in S,\ c, c' \in \mathcal{C}(A,k),\ \gamma \in \mathcal{A}(A, k-1)\big\} & \qquad (5)
\end{aligned}$$

We still have to establish that G' indeed generates exactly $\mathcal{R}(G)$. This will be achieved by showing both inclusions in the next chain of lemmas.

▶ **Lemma 18.** $\mathcal{F}(G') \subseteq \mathcal{R}(G)$*.*

For the converse, we decompose and encode rule trees $\mathcal{R}(G)$ in a more compact manner. First, we translate a rule tree into its *primary spine form*. For all $a \in A$, $c \in \mathcal{C}(A,k)$, $\gamma \in \mathcal{A}(A,k)$, and $t_1, t_2 \in \mathrm{T}$ we let

$$\begin{aligned}
\mathrm{spinal}\big(c\big) &= c\\
\mathrm{spinal}\big((ax/c,\ c\gamma \to ax\gamma)(t_1, t_2)\big) &= (ax/c,\ c\gamma \to ax\gamma)\big(\mathrm{spinal}(t_1), \langle c\rangle\big)\\
\mathrm{spinal}\big((c\gamma,\ ax\backslash c \to ax\gamma)(t_1, t_2)\big) &= (c\gamma,\ ax\backslash c \to ax\gamma)\big(\langle c\rangle, \mathrm{spinal}(t_2)\big) \enspace .
\end{aligned}$$

Clearly, $\mathrm{spinal}\colon \mathrm{T} \to T_{R,\emptyset}(L(\Sigma) \cup N_0)$. An example is shown in Figure 5. Additionally, we encode rule trees using only the nonterminals of G' [i.e. a tree of $T_{\emptyset,N_1}(N_0)$]. To this end, we define a mapping $\mathrm{enc}\colon \mathrm{T} \to T_{\emptyset,N_1}(N_0)$. For all $a \in A$, $c \in \mathcal{C}(A,k)$, $\gamma \in \mathcal{A}(A,k)$, and $t_1, t_2 \in \mathrm{T}$ we let

$$\begin{aligned}
\mathrm{enc}(c) &= \langle c\rangle\\
\mathrm{enc}\big((ax/c,\ c\gamma \to ax\gamma)(t_1, t_2)\big) &= \langle a, /c, \gamma\rangle\big(\mathrm{enc}(t_1)\big)\\
\mathrm{enc}\big((c\gamma,\ ax\backslash c \to ax\gamma)(t_1, t_2)\big) &= \langle a, \backslash c, \gamma\rangle\big(\mathrm{enc}(t_2)\big) \enspace .
\end{aligned}$$

This encoding is also demonstrated in Figure 5.

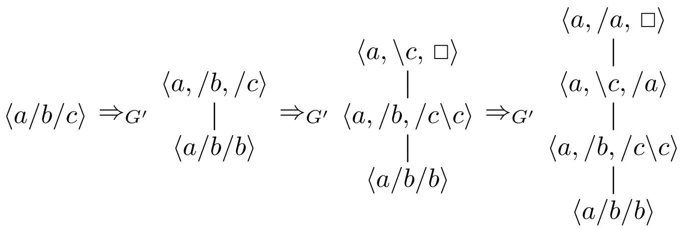

$\langle a/b/c\rangle \Rightarrow_{G'}$
$\langle a, /b, /c\rangle$
$\langle a/b/b\rangle$
$\Rightarrow_{G'}$
$\langle a, \backslash c, \Box\rangle$
$\langle a, /b, /c\backslash c\rangle$
$\langle a/b/b\rangle$
$\Rightarrow_{G'}$
$\langle a, /a, \Box\rangle$
$\langle a, \backslash c, /a\rangle$
$\langle a, /b, /c\backslash c\rangle$
$\langle a/b/b\rangle$

Figure 6 Derivation of the encoding.

▶ **Lemma 19.** $\langle \mathrm{cat}_G(t)\rangle \Rightarrow^*_{G'} \mathrm{enc}(t) \Rightarrow^*_{G'} \mathrm{spinal}(t)$ *for every* $t \in \mathrm{T}$ *with* $\mathrm{cat}_G(t) \in \mathcal{C}(A, k)$, *and hence* $\mathcal{R}(G) \subseteq \mathcal{F}(G')$.

▶ **Theorem 20.** *The rule trees* $\mathcal{R}(G)$ *of a CCG* G *can be generated by an sCFTG.*

7 Proper Inclusion for Pure CCGs

In this section, we show that there exist CFG derivation tree languages that cannot be accepted by any pure CCG. A CCG (Σ, A, R, I, L) is *pure* if $R = \mathcal{R}(A, k)$ for some $k \in \mathbb{N}$. In particular, this shows that the inclusion demonstrated in Section 6 is proper for pure CCGs. We start with our counterexample CFG. To make the text more readable, we assume henceforth that all computations with nonterminals are performed modulo 3.

▶ **Example 21.** Let us consider the CFG $G_{\mathrm{ex}} = (N, \Gamma, \langle 0, 0\rangle, P)$ with the nonterminals $N = \{\langle i, j\rangle \mid 0 \le i, j \le 2\}$, the terminals $\Gamma = \{\alpha\}$, and the set P of productions contains exactly $\langle i, j\rangle \to \langle i+1, j\rangle\ \langle i, j+1\rangle$ and $\langle i, j\rangle \to \alpha$ for every $\langle i, j\rangle \in N$. Clearly, the tree language $\mathcal{D}(G_{\mathrm{ex}})$ is not universally mht-bounded.

Theorem 9 already shows that the tree language $\mathcal{D}(G_{\mathrm{ex}})$ cannot be accepted by any 0-CCG. Similarly, it is impossible to accept $\mathcal{D}(G_{\mathrm{ex}})$ with a pure CCG. This follows from the transformation schemes of [14] that change the order of consecutive application and non-application operations, resulting in derivation trees with reordered subtrees and therefore with the wrong shape after relabeling. Due to the absence of rule restrictions in pure CCGs, the applicability of these transformations cannot be prevented.

▶ **Theorem 22.** *The tree language* $\mathcal{D}(G_{\mathrm{ex}})$ *cannot be accepted any pure CCG.*

8 Conclusion

We have shown that the tree languages accepted by CCGs with limited composition depth and rule restrictions are a subset of the tree languages generated by simple monadic context-free tree grammars. This inclusion is proper for pure CCGs (i.e. without rule restrictions). In addition, the tree languages accepted by 0-CCGs are a proper subset of regular tree languages, whereas those accepted by 1-CCGs are exactly the regular tree languages. While the step from 0-CCGs to 1-CCGs does not increase the weak generative capacity, the strong generative capacity increases. We also observe that there is no difference in expressivity for 0-CCGs between the pure and non-pure variants, while for higher rule degrees, pure CCGs are strictly weaker. The first statement follows from the fact that we are able to construct an equivalent pure 0-CCG for each mht-bounded regular tree language.

The construction used in the classical proof of weak equivalence between CCG and TAG [28] demonstrated that there is no difference in weak generative capacity between 2-CCGs and k-CCGs with $k > 2$ and that the inclusion of higher-order categories in the lexicon does not change weak generative capacity. However, as stated in the Introduction, this construction utilizes ε-entries, which are problematic from a computational point of view [15]. Future work should explore the relationship between TAG and CCG, and in particular the effects of higher rule degrees k (up to unlimited composition depth) and higher-order categories, in the absence of ε-entries. Another interesting direction is strong generative capacity: If for a given sCFTG a strongly equivalent CCG could be constructed (the inverse direction of what we showed in Theorem 20), this would characterize the tree-generative capacity of CCG exactly. Furthermore, the effect of the inclusion of additional rules on the expressivity should be studied.

References

1 Kazimierz Ajdukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

3 Yehoshua Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29(1):47–58, 1953.

4 Yehoshua Bar-Hillel, Haim Gaifman, and Eli Shamir. On Categorial and Phrase Structure Grammars. In Yehoshua Bar-Hillel, editor, *Language and Information: Selected Essays on Their Theory and Application*, pages 99–115. Addison Wesley, 1964.

5 Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. In Yehoshua Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison Wesley, 1964.

6 Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

7 Haskell B. Curry. Foundations of combinatorial logic. *American Journal of Mathematics*, 52(3):509–536, 1930.

8 Haskell B. Curry, Robert Feys, and William Craig. *Combinatory Logic*. Number 1 in Studies in Logic and the Foundations of Mathematics. North-Holland, 1958.

9 Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984. 2nd revision available at `arXiv:1509.06233`.

10 Ferenc Gécseg and Magnus Steinby. Tree Languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer, 1997.

11 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

12 Aravind K. Joshi and Yves Schabes. Tree-Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Beyond Words*, volume 3 of *Handbook of Formal Languages*, pages 69–123. Springer, 1997.

13 Alexander Koller and Marco Kuhlmann. Dependency Trees and the Strong Generative Capacity of CCG. In *Proc. 12th EACL*, pages 460–468. ACL, 2009.

14 Marco Kuhlmann, Alexander Koller, and Giorgio Satta. Lexicalization and Generative Power in CCG. *Comput. Linguist.*, 41(2):187–219, 2015.

15 Marco Kuhlmann, Giorgio Satta, and Peter Jonsson. On the Complexity of CCG Parsing. *Comput. Linguist.*, 44(3):447–482, 2018.

16 Joachim Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65(3):154–170, 1958.

17 Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Global neural CCG parsing with optimality guarantees. In *Proc. 2016 EMNLP*, pages 2366–2376. ACL, 2016.

18 Mike Lewis and Mark Steedman. Unsupervised induction of cross-lingual semantic relations. In *Proc. 2013 EMNLP*, pages 681–692. ACL, 2013.

19 William C. Rounds. Context-Free Grammars on Trees. In *Proc. 1st STOC*, pages 143–148. ACM, 1969.

20 William C. Rounds. Tree-Oriented Proofs of Some Theorems on Context-Free and Indexed Languages. In *Proc. 2nd STOC*, pages 109–116. ACM, 1970.

21 Yves Schabes, Anne Abeillé, and Aravind K. Joshi. Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In *Proc. 12th CoLing*, pages 578–583, 1988.

22 Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92(3–4):305–316, 1924.

23 Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

24 Mark Steedman and Jason Baldridge. Combinatory Categorial Grammar. In Robert D. Borsley and Kersti Börjars, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*, chapter 5, pages 181–224. Blackwell, 2011.

25 Hans-Jörg Tiede. *Deductive Systems and Grammars: Proofs as Grammatical Structures*. PhD thesis, Indiana University, Bloomington, IN, USA, 1999.

26 Krishnamurti Vijay-Shanker and David J. Weir. Combinatory Categorial Grammars: Generative Power and Relationship to Linear Context-Free Rewriting Systems. In *Proc. 26th ACL*, pages 278–285. ACL, 1988.

27 Krishnamurti Vijay-Shanker and David J. Weir. Polynomial time parsing of combinatory categorial grammars. In *Proc. 28th ACL*, pages 1–8. ACL, 1990.

28 Krishnamurti Vijay-Shanker and David J. Weir. The Equivalence of Four Extensions of Context-Free Grammars. *Math. Systems Theory*, 27(6):511–546, 1994.

Cyclic Proofs and Jumping Automata

Denis Kuperberg
Univ Lyon, CNRS, ENS de Lyon, UCBL, LIP UMR 5668, F-69342, LYON Cedex 07, France

Laureline Pinault
Univ Lyon, CNRS, ENS de Lyon, UCBL, LIP UMR 5668, F-69342, LYON Cedex 07, France

Damien Pous
Univ Lyon, CNRS, ENS de Lyon, UCBL, LIP UMR 5668, F-69342, LYON Cedex 07, France

Abstract

We consider a fragment of a cyclic sequent proof system for Kleene algebra, and we see it as a computational device for recognising languages of words. The starting proof system is linear and we show that it captures precisely the regular languages. When adding the standard contraction rule, the expressivity raises significantly; we characterise the corresponding class of languages using a new notion of multi-head finite automata, where heads can jump.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Cyclic proofs, regular languages, multi-head automata, transducers

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.45

Related Version Appendix with proofs available at `https://hal.archives-ouvertes.fr/hal-02301651`.

Funding This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157), and was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

1 Introduction

Cyclic proof systems have received much attention in the recent years. Proofs in such systems are graphs rather than trees, and they must satisfy a certain validity criterion.

Such systems have been proposed for instance by Brotherston and Simpson in the context of first order logic with inductive predicates [4], as an alternative to the standard induction schemes. The infinite descent principles associated to cyclic proofs are in general at least as powerful as the standard induction schemes, but the converse is a delicate problem. It was proven only recently that it holds in certain cases [3, 18], and that there are also cases where cyclic proofs are strictly more expressive [2].

Cyclic proof systems have also been used in the context of the μ-calculus [8], where we have inductive predicates (least fixpoints), but also coinductive predicates (greatest fixpoints), and alternation of those. Proof theoretical aspects such as cut-elimination were studied from the linear logic point of view [11, 10], and these systems were recently used to obtain constructive proofs of completeness for Kozen's axiomatisation [9, 1].

Building on these works, Das and Pous considered the simpler setting of Kleene algebra, and proposed a cyclic proof system for regular expression containments [6]. The key observation is that regular expressions can be seen as μ-calculus formulas using only a single form of fixpoint: the definition of Kleene star as a least fixpoint ($e^* = \mu x.1 + e \cdot x$). Their system is based on a non-commutative version of μMALL [10], and it is such that a sequent $e \vdash f$ is derivable if and only if the language of e is contained in that of f. This work eventually led to an alternative proof of left-handed completeness for Kleene algebra [5].

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 45; pp. 45:1–45:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the latter works, it is natural to consider regular expressions as datatypes [12], and proofs of language containments as total functions between those datatypes [13]. Such a computational interpretation of cyclic proofs was exploited to prove cut-elimination in [7].

We follow the same approach here, focusing on an even simpler setting: our sequents essentially have the shape $A^* \vdash 2$, where A is a finite alphabet and 2 is a type (or formula) for Boolean values. Cyclic proofs no longer correspond to language containments: they give rise to functions from words to Booleans, *i.e.* , formal languages. We characterise the class of languages that arise from such proofs.

If we keep a purely linear proof system, as in [6, 7], we show that we obtain exactly the regular languages. In contrast, if we allow the contraction rule, we can express non-regular languages. We show that in this case, we obtain the languages that are recognisable by a new class of automata, which we call *jumping multihead automata*[1]. Indeed, cyclic proofs are more expressive than the plain one-way multihead automata that were studied in the literature [14]. Intuitively, when reading a word, a multihead automaton may only move its heads forward, letter by letter, while a jumping multihead automaton also has the possibility to let a given head jump to the position of another head. This gives the opportunity to record positions in the word, and to repeatedly analyse the suffixes starting from those positions.

Outline

We define our cyclic proof system and its computational interpretation in Sect. 2. Then we define jumping multihead automata and establish their basic properties (Sect. 3). We prove the equivalence between the two models in Sect. 4 (Thm. 19), from which it follows that we capture precisely the regular languages in the linear case (Thm. 24). We discuss directions for future work in Sect. 5.

Notation

Given sets X, Y, we write $X \times Y$ for their Cartesian product, $X \uplus Y$ for their disjoint union, and X^* for the set of finite sequences (lists) over X. Given such a sequence l, we write $|l|$ for its length and l_i or $l(i)$ for its i^{th} element. We write $\mathbb{B}$ for the set $\{\mathbf{ff}, \mathbf{tt}\}$ of Booleans, and $\langle x, y, z\rangle$ for tuples. We use commas to denote concatenation of both sequences and tuples, and ϵ to denote the empty sequence. We write $Im(f)$ for the image of a function f.

2 Infinite proofs and their semantics

We let a, b range over the letters of a fixed and finite alphabet A. We work with only two *types* (or *formulas*): the type A of letters, and the type A^* of words. We let e, f range over types, and E, F range over finite sequences of types. Given such a sequence $E = e_1, \ldots, e_n$, we write $[E]$ for the set $e_1 \times \cdots \times e_n$.

We define a sequent proof system, where sequents have the shape $E \vdash 2$, and where proofs of such sequents denote functions from $[E]$ to $\mathbb{B}$, *i.e.* subsets of $[E]$.

[1] This new class should not be confused with the *jumping finite automata* introduced by Meduna and Zemek [16], which are not multihead.

$$w\ \frac{E,F\vdash 2}{E,e,F\vdash 2} \qquad c\ \frac{E,e,e,F\vdash 2}{E,e,F\vdash 2} \qquad A\ \frac{(E,F\vdash 2)_{a\in A}}{E,A,F\vdash 2} \qquad *\ \frac{E,F\vdash 2 \qquad E,A,A^*,F\vdash 2}{E,A^*,F\vdash 2} \qquad t\ \frac{}{\vdash 2} \qquad f\ \frac{}{\vdash 2}$$

Figure 1 The rules of C.

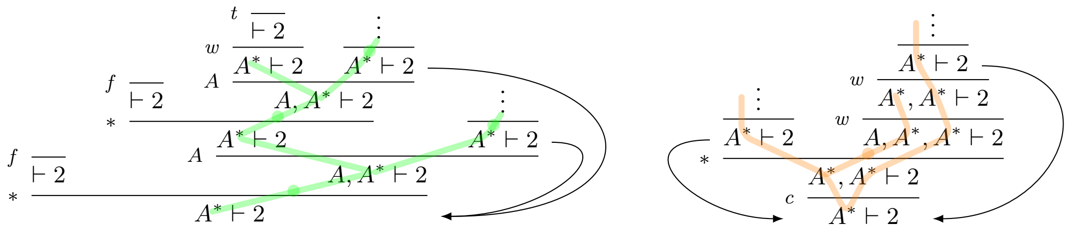


Figure 2 Two regular preproofs; only the one on the left is valid.

2.1 Infinite proofs

We now define the cyclic proof system whose six inference rules are given in Fig. 1. In addition to two *structural rules* (weakening and contraction), we have a left introduction rule for each type, and two right introduction rules for Boolean constants. Note that there is no exchange rule, which explains why the structural and left introduction rules use two sequences E and F rather than a single one.

The left introduction rule for type A^* corresponds to an unfolding rule, looking at A^* as the least fixpoint expression $\mu X.(1 \uplus A \times X)$ (e.g., from μ-calculus). The left premiss intuitively corresponds to the case of an empty list, while the right premiss covers the case of a non-empty list. Except from weakening and contraction, those rules form a very small fragment of those used for Kleene algebra in [7] (interpreting A as a sum $1+\cdots+1$ with $|A|$ elements and 2 as the binary sum $1+1$).

Note that we are not interested in *provability* in the present paper: every sequent can be derived trivially, using weakenings and one of the two right introduction rules. The objects of interest are the proofs themselves; this explains why we have two axioms for proving the sequent $\vdash 2$: they correspond to two different proofs.

We set $B = A \uplus \{0,1\}$. A (possibly infinite) *tree* is a non-empty and prefix-closed subset of B^*, which we view with the root, ϵ, at the bottom; elements of B^* are called *addresses.*

▶ **Definition 1.** *A* preproof *is a labelling π of a tree by sequents such that, for every node v with children $v_1,\dots v_n$, the expression $\dfrac{\pi(v_1)\ \cdots\ \pi(v_n)}{\pi(v)}$ is an instance of a rule from Fig. 1. A preproof is* regular *if it has finitely many distinct subtrees,* i.e. *it can be viewed as the unfolding of a finite graph. A preproof is* affine *if it does not use the c-rule.*

If π is a preproof, we note $Addr(\pi)$ its set of addresses, *i.e.* its underlying tree. The formulas appearing in lists E, F of any rule instance are called *auxiliary formulas.* The non auxiliary formula appearing in the conclusion of a rule is called the *principal formula.*

A $*$ *address* in a preproof π is an address v which is the conclusion of a $*$ rule in π.

Two examples of regular preproofs are depicted in Fig. 2. The alphabet A is assumed to have exactly two elements, so that the A rule is binary. Backpointers are used to denote circularity: the actual preproofs are obtained by unfolding the graphs. The preproof on the right might look suspicious: it never uses the axioms t or f. In fact, only the one on the left satisfies the validity criterion which we define below. Before doing so, we need to define a notion of thread, which are the branches of the shaded trees depicted on the preproofs. Intuitively a thread follows a star formula occurrence along a branch of the proof. First we need to define parentship and ancestor relations.

▶ **Definition 2.** *A* [star] position *in a preproof π is a pair $\langle v, i\rangle$ consisting of an address v and an index $i \in [0; |E| - 1]$, where $\pi(v) = E \vdash 2$ [and E_i is a star formula]. A position $\langle w, j\rangle$ is the* parent *of a position $\langle v, i\rangle$ if $|v| = |w| + 1$ and, looking at the rule applied at address w the two positions point at the same place in the lists E, F of auxiliary formulas, or at the formula e when this is the contraction rule, or at the principal formula A^* when this is the $*$ rule and $v = w1$. We write $\langle v, i\rangle \lhd \langle w, j\rangle$ in the former cases, and $\langle v, i\rangle \mathrel{\dot{\lhd}} \langle w, j\rangle$ in the latter case. Position $\langle w, j\rangle$ is an* ancestor *of $\langle v, i\rangle$ when those positions are related by the transitive closure of the parentship relation.*

The graph of the parentship relation is depicted in Fig. 2 using shaded thick lines and an additional bullet to indicate when we pass principal star steps ($\dot{\lhd}$). Note that in the $*$ rule, the principal formula occurrence A^* is not considered as a parent of the occurrence of A in the right premiss.

We can finally define threads and the validity criterion.

▶ **Definition 3.** *A* thread *is a possibly infinite branch of the ancestry graph. A thread is* principal *when it visits a $*$ rule through its principal formula. A thread is* valid *if it is principal infinitely often.*

In the first preproof of Fig. 2, the infinite green thread $\langle \epsilon, 0\rangle \mathrel{\dot{\rhd}} \langle 1, 1\rangle \rhd \langle 11, 0\rangle \mathrel{\dot{\rhd}} \langle 111, 1\rangle \rhd \langle 1111, 0\rangle \ldots$ is valid, as well as every other infinite thread. There is no valid thread in the second preproof: taking a principal step forces the thread to terminate.

▶ **Definition 4.** *A preproof is* valid *if every infinite branch contains a valid thread. A* proof *is a valid preproof. We write $\pi : E \vdash 2$ when π is a proof whose root is labelled by $E \vdash 2$.*

In the examples from Fig. 2, only the preproof on the left is valid, thanks to the infinite green thread. The second preproof is invalid: infinite threads along the (infinitely many) infinite branches are never principal.

This validity criterion is essentially the same as for the system LKA [7], which in turn is an instance of the one used for μMALL [10]: we just had to extend the notion of ancestry to cover the contraction rule. Note however that the presence of this rule induces some subtleties. For instance, while in the cut-free fragment of LKA, a preproof is valid if and only if it is *fair* (*i.e.* every infinite branch contains infinitely many $*$ steps [7, Prop. 8]), this is no longer true with contraction: the second preproof from Fig. 2 is fair and invalid.

In the affine case, due to the fragment we consider here, and since we do not include cut, the situation is actually trivial:

▶ **Proposition 5.** *Every affine preproof is valid.*

2.2 Computational interpretation of infinite proofs

We now show how to interpret a proof $\pi : E \vdash 2$ as a function $[\pi] : [E] \to \mathbb{B}$. Since proofs are not well-founded, we cannot reason directly by induction on proofs. We use instead the following relation on partial computations, which we prove to be well-founded thanks to the validity criterion.

▶ **Definition 6.** *A* partial computation *in a fixed proof π is a pair $\langle v, s\rangle$ consisting of an address v of π with $\pi(v) = E \vdash 2$, and a value $s \in [E]$*

Given two partial computations, we write $\langle v, s\rangle \prec \langle w, t\rangle$ when

1. $|v| = |w| + 1$,

2. *for every i, j such that $\langle v, i\rangle \lhd \langle w, j\rangle$, we have $s_i = t_j$, and*

3. *for every i, j such that $\langle v, i\rangle \mathrel{\dot{\lhd}} \langle w, j\rangle$, we have $|s_i| < |t_j|$.*

The first condition states that the subproof at address v should be one of the premisses of the subproof at w; the second condition states that the values assigned to star formulas should remain the same along auxiliary steps; the third condition ensures that they actually decrease in length along principal steps.

▶ **Lemma 7.** *The relation $\prec$ on partial computations is well-founded.*

Proof. Suppose by contradiction that there exists an infinite descending sequence. By condition 1/, this sequence corresponds to an infinite branch of π. By validity, this branch must contain a thread which is principal infinitely many times. This thread contradicts conditions 2/ and 3/ since we would obtain an infinite sequence of lists of decreasing length. ◀

▶ **Definition 8.** *The* return value $[v](s)$ *of a partial computation $\langle v, s\rangle$ with $\pi(v) = E \vdash 2$ is a Boolean defined by well-founded induction on $\prec$ and case analysis on the rule used at address v.*

$$\begin{array}{rl@{\qquad}rl}
w: & [v](s, x, t) \triangleq [v0](s, t) & A: & [v](s, a, t) \triangleq [va](s, t) \\
c: & [v](s, x, t) \triangleq [v0](s, x, x, t) & *: & [v](s, l, t) \textit{ is defined by case analysis on the list } l\textit{:} \\
t: & [v]() \triangleq \mathbf{tt} & & - \ [v](s, \epsilon, t) \triangleq [v0](s, t) \\
f: & [v]() \triangleq \mathbf{ff} & & - \ [v](s, x :: q, t) \triangleq [v1](s, x, q, t)
\end{array}$$

In each case, the recursive calls are made on strictly smaller partial computations: they occur on direct subproofs, the values associated to auxiliary formulas are left unchanged, and in the second subcase of the $*$ case, the length of the list associated to the principal formula decreases by one.

▶ **Definition 9.** *The semantics of a proof $\pi : E \vdash 2$ is the function $[\pi] \colon s \mapsto [\epsilon](s)$.*

(Note that we could give a simpler definition of the semantics for affine proofs by reasoning on the total size of the arguments; such an approach however breaks in presence of contraction.)

Let us compute the semantics of the first (and only) proof in Fig. 2. Recall that A has two elements in this example, so set $A = \{a, b\}$ (and thus $B = \{0, 1, a, b\}$), and let us use a (resp. b) to navigate to the left (resp. right) premiss of the A rule. Starting from words aba and aab, we get the two computations on the left below:

$$\begin{array}{rl@{\qquad\qquad}rl@{\qquad\qquad}rcl}
 & [\epsilon](ab) & & [\epsilon](aab) & [\epsilon](\epsilon) & = & \mathbf{ff} \\
= & [1](a, b) & = & [1](a, ab) & [\epsilon](au) & = & [\epsilon](u) \\
= & [1a](b) & = & [1a](ab) & [\epsilon](bu) & = & [1a](u) \\
= & [1a1](b, \epsilon) & = & [1a1](a, b) & & & \\
= & [1a1b](\epsilon) & = & [1a1a](b) & [1a](\epsilon) & = & \mathbf{ff} \\
= & [1a1b0]() & = & [1a1a0]() & [1a](au) & = & \mathbf{tt} \\
= & \mathbf{ff} & = & \mathbf{tt} & [1a](bu) & = & [\epsilon](u)
\end{array}$$

Using the fact that the subproofs at addresses ϵ, $1a$ and $1a1b$ are identical, we can also deduce the equations displayed on the right, which almost correspond to the transition table of a deterministic automaton with two states ϵ and $1a$. This is not strictly speaking a

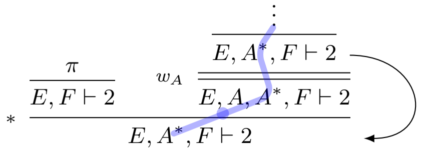


Figure 3 Weakening stars (Prop. 10).

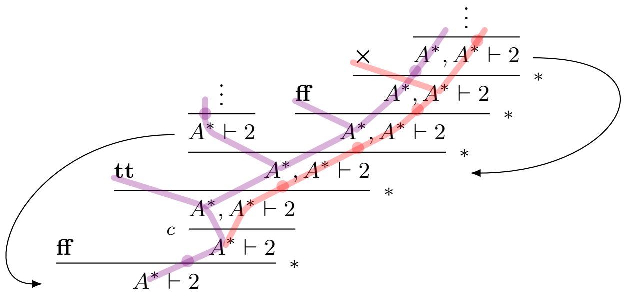


Figure 4 A regular proof for $\{a^{2^n} \mid n \in \mathbb{N}\}$.

deterministic automaton because of the fifth line: when reading an a, the state $1a$ decides to accept immediately, whatever the remainder of the word. We can nevertheless deduce from those equations that ϵ recognises the language A^*aaA^*.

Trying to perform such computations on the invalid preproof on the right in Fig. 2 gives rise to non-terminating behaviours, e.g., $[\epsilon](\epsilon) \rightsquigarrow [0](\epsilon, \epsilon) \rightsquigarrow [00](\epsilon) \rightsquigarrow \dots$ and $[\epsilon](x :: q) \rightsquigarrow [0](x :: q, x :: q) \rightsquigarrow [01](x, q, x :: q) \rightsquigarrow [010](q, x :: q) \rightsquigarrow [0100](x :: q) \rightsquigarrow \dots$.

Before studying a more involved example, we prove the following property:

▶ **Proposition 10.** *The weakening rule* (w) *is derivable in a way that respects regularity, affinity, existing threads, and the semantics.*

Proof. When the weakened formula is A, it suffices to apply the A rule and to use the starting proof $|A|$ times. When the weakened formula is A^*, assuming a proof $\pi : E, F \vdash 2$, we construct the proof in Fig. 3. The step marked with w_A is the previously derived weakening on A. The preproof is valid because this step does preserve the blue thread. ◀

As a consequence, the full proof system is equivalent to the one without weakening. We shall see that the system would remain equally expressive with the addition of an exchange rule (see Rem. 23 below), but that the contraction rule instead plays a crucial role and changes the expressive power.

Let us conclude this section with an example beyond regular languages: we give in Fig. 4 a proof whose semantics is the language of words over a single letter alphabet, whose length is a power of two (a language which is not even context-free). Since the alphabet has a single letter, the A rule becomes a form of weakening, and we apply it implicitly after each $*$ step. We also abbreviate subproofs consisting of a sequence of weakenings followed by one of the two axioms by **tt**, **ff**, or just $\times$ when it does not matter whether we return true or false.

Writing n for the word of length n and executing the proof on small numbers, we observe

$$\begin{aligned}
&[\epsilon](0) = [0]() = \mathbf{ff}\\
&[\epsilon](1) = [1](0) = [10](0,0) = [100](0) = \mathbf{tt}\\
&[\epsilon](2) = [1](1) = [10](1,1) = [101](1,0) = [1010](1) = [\epsilon](1) = \mathbf{tt}\\
&[\epsilon](3) = [1](2) = [10](2,2) = [101](2,1) = [1011](2,0) = \mathbf{ff}\\
&[\epsilon](4) = [1](3) = [10](3,3) = [101](3,2) = [1011](3,1) = [10111](3,0) = [101111](2,0)\\
&\qquad = [101](2,0) = [1010](2) = [\epsilon](2) = \mathbf{tt}
\end{aligned}$$

More generally, the idea consists in checking that the given number can be divided by two repeatedly, until we get 1. To divide a number represented in unary notation by two, we copy that number using the contraction rule, and we consume one of the copies twice as fast as the other one (through the three instances of the $*$ rule used at addresses 101, 1011, and 10111); if we reach the end of one copy, then the number was even, the other copy precisely contains its half, and we can proceed recursively (through the backpointer on the left), otherwise the number was odd and we can reject. The subproof at address 101110 is never explored: we would be in a situation where the slowly consumed copy gets empty before the other one.

Finally note that every (even undecidable) language can be represented using an infinite (in general non regular) proof: apply the left introduction rules eagerly, and fill in the left premisses of the $*$ rules using the appropriate axiom.

3 Jumping multihead automata

Now we introduce the model of Jumping Multihead Automata (JMA) and establish its basic properties. We will prove in Sect. 4 that its expressive power is precisely that of cyclic proofs.

3.1 Definition and semantics of JMAs

Let A be a finite alphabet and $\lhd \notin A$ be a fresh symbol. We note $A_\lhd = A \cup \{\lhd\}$.

▶ **Definition 11.** *A jumping multihead automaton (JMA) is a tuple $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta\rangle$ where:*

- *S is a finite set of states;*
- *$k \in \mathbb{N}$ is the number of heads;*
- *$s_0 \in S$ is the initial state;*
- *$s_{acc} \in S$ and $s_{rej} \in S$ are final states, respectively accepting and rejecting;*
- *$\delta : S_{trans} \times (A_\lhd)^k \longrightarrow S \times Act^k$ is the deterministic transition function, where $S_{trans} \triangleq S \setminus \{s_{acc}, s_{rej}\}$ is the set of non-final states, and $Act \triangleq \{\text{◌}, \blacktriangleright\!|\} \cup \{J_1, J_2, \ldots, J_k\}$.*

In the transition function, symbols ◌ and $\blacktriangleright\!|$ stand for "stay in place" and "move forward" respectively, and action J_i stands for "jump to the position of head number i". Intuitively, if the machine is in state s, each head j reads letter $\vec{a}(j)$, and $\delta(s, \vec{a}) = (s', \alpha)$, then the machine goes to state s' and each head j performs the action $\alpha(j)$. Accordingly, to guarantee that the automaton does not try to go beyond the end marker of the word, we require that if $\delta(s, \vec{a}) = (s', \alpha)$, then for all $j \in [\![1, k]\!]$ with $\vec{a}(j) = \lhd$ we have $\alpha(j) \neq \blacktriangleright\!|$.

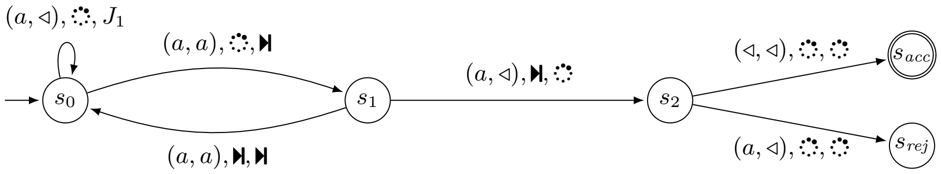
A *configuration* of a JMA $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta\rangle$ is a triple $c = (w, s, p)$ where w is the input word, $s \in S$ is the current state, and $p = (p_1, \ldots, p_k) \in [\![0, |w|]\!]^k$ gives the current head positions. If the position p_i is $|w|$ then the head i is scanning the symbol $\lhd$.

The initial configuration on an input word w is $(w, s_0, (0, \ldots, 0))$. Let $w = a_0 a_1 \ldots a_{n-1}$ be the input and $a_n = \lhd$. Let $(w, s, (p_1, \ldots, p_k))$ be a configuration with $s \in S_{trans}$, and $(s', (x_1, \ldots, x_k)) = \delta(s, (a_{p_0}, \ldots a_{p_k}))$ be given by the transition function. Then the successor configuration is defined by $(w, s', (p'_1, \ldots, p'_k))$, where for all $i \in [\![1, k]\!]$ p'_i depends on x_i in the following way:

(1) $p'_i = p_i$ if $x_i = \text{◌}$ (2) $p'_i = p_i + 1$ if $x_i = \blacktriangleright\!|$ (3) $p'_i = p_j$ if $x_i = J_j$

A configuration (w, s, p) is *final* if $s \in \{s_{acc}, s_{rej}\}$. It is accepting (resp. rejecting) if $s = s_{acc}$ (resp. $s = s_{rej}$). A *run* of a JMA $\mathcal{M}$ on w is a sequence of configurations $c_0, c_1, \ldots, c_r$ on w where c_0 is the initial configuration, and c_{i+1} is the successor configuration of c_i for all i. If c_r is rejecting (resp. accepting), we say that the word w is rejected (resp. accepted) by $\mathcal{M}$. We say that $\mathcal{M}$ *terminates* on w if there is a maximal finite run of $\mathcal{M}$ on w, ending in a final configuration. The *language* of $\mathcal{M}$, noted $L(\mathcal{M})$, is the set of finite words accepted by $\mathcal{M}$, *i.e.* the set of words $w \in A^*$ such that $\mathcal{M}$ has an accepting run on w.

▶ **Example 12.** The language $L = \{a^{2^n} \mid n \in \mathbb{N}\}$ can be recognised by the following JMA with two heads. (Missing transitions all go to the rejecting final state.)



The idea behind the automaton is similar as the proof given in Fig. 4: one head advances at twice the speed of the other. When the fast head reaches the end of the word, it either rejects if the length is odd and at least 2, or jumps to the position of the slow head located in the middle of the word. From there, the automaton proceeds recursively.

Notice that on an input word u, three scenarios are possible: the automaton accepts by reaching s_{acc}, rejects by reaching s_{rej}, or rejects by looping forever. In order to translate JMAs into cyclic proofs, whose validity criterion ensures termination, it is convenient to forbid the last scenario. We ensure such a property by a syntactic restriction on the transition structure of JMAs.

The *transition graph* of a JMA $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta \rangle$ is the labelled graph $G_{\mathcal{M}} = (S, E)$, where the vertices are states S, and the set of edges is $E \subseteq S \times S \times Act^k$, defined by $E = \{(s, s', \alpha) \mid \exists \vec{a} \in (A_{\lhd})^k, \delta(s, \vec{a}) = (s', \alpha)\}$.

A JMA $\mathcal{M}$ is *progressing* if for every cycle $e_1 e_2 \dots e_l$ in its transition graph, where $e_i = (s_i, s_{i+1}, \alpha_i)$ for each $i \in [\![1, l]\!]$ and $s_{l+1} = s_1$, there exists a head $j \in [\![1, k]\!]$ with $\alpha_1(j)\alpha_2(j) \dots \alpha_l(j) \in (◌^* \cdot ⏭ \cdot ◌^*)^+$. (Intuitively we require that for every loop, one of the head does not jump during this loop and moves forward at least once).

The JMA from Ex. 12 always terminates, but it is not progressing due to the loop on the initial state. It could easily be modified into a progressing JMA by introducing a new intermediary state instead of looping on s_0. In fact, even in cases where a JMA can indefinitely loop on some inputs, one can always turn it into a progressing one recognising the same language. Hence all JMAs are assumed to be progressing from now on.

▶ **Lemma 13.** *Every JMA can be converted into a progressing JMA with the same language.*

Proof. We use the fact that the number of possible configurations on a given word w is bounded polynomially in the length of w. We add heads to the JMA that just advance counting up until this bound, making the JMA progressing. Details are given in [15, Appendix]. ◀

▶ **Lemma 14.** *Given a JMA $\mathcal{M}$, we can check in* NL *whether $\mathcal{M}$ is progressing. If $\mathcal{M}$ is progressing, then it terminates on all words.*

3.2 Expressive power of JMAs

Write *JMA*(k) for the set of languages expressible by a progressing JMA with k heads. JMAs encode only DLogSpace languages; one-head JMAs capture exactly the regular languages.

▶ **Lemma 15.** $\bigcup_{k \geq 1} JMA(k) \subseteq$ DLogSpace.

Proof. It is straightforward to translate a JMA with k heads into a Turing machine using space $O(\log^k(n))$, by remembering the position of the heads. ◀

▶ **Lemma 16.** *$JMA(1) = \textsc{Reg}$.*

As mentioned in the introduction, (non-jumping) multihead automata have already been investigated in the literature [14]. They consist of automata with a fixed number of heads (k) that can either only go from left to right, (like our JMAs, case of 1-way automata, $1DFA(k)$), or in both directions (case of 2-ways automata, $2DFA(k)$). We briefly compare JMAs to those automata, starting with the 1-way case.

First of all, it is clear that for all $k \geq 1$, $1DFA(k) \subseteq JMA(k)$ (in particular, because $1DFA$s can be assumed to be progressing without increasing the number of heads).

▶ Remark 17. Since emptiness, universality, regularity, inclusion and equivalence are undecidable for $1DFA$s with 2 heads [14], these problems are also undecidable for JMAs with 2 heads.

The following proposition shows that the ability to jump increases the expressive power.

▶ **Proposition 18.** *For all $k \geq 1$, $JMA(2) \nsubseteq 1DFA(k)$.*

Proof. It is proven in [19] that $(1DFA(k))_{k\in\mathbb{N}}$ forms a strict hierarchy, by defining a language L_b that is recognisable by a $1DFA$ with k heads if and only if $b < \binom{k}{2}$. We slightly modify these languages so that they become expressible with a two heads JMA while keeping the previous characterisation for $1DFA$. Details are given in [15, Appendix]. ◀

Concerning 2-ways automata ($2DFA$) it is known that $\bigcup_{k\geq 1} 2DFA(k) = \textsc{DLogSpace}$ [14], so that by Lem. 15 every JMA can be translated into a deterministic multihead 2-way automaton, not necessarily preserving the number of heads. The converse direction is more delicate. The language of palindromes belongs to $2DFA(2)$, but we conjecture that it cannot be represented by a JMA, whatever the number of heads. We also conjecture that $(JMA(k))_{k\in\mathbb{N}}$ forms a strict hierarchy: we think that the language $L = a^{l_1}\$a^{l_2}\$\dots\$a^{l_k}\$a^{\Pi_{i=1}^{k} l_i}$ can be recognised by a JMA only if it has strictly more than k heads.

4 Equivalence between JMAs and cyclic proofs

We now turn to proving the following characterisation.

▶ **Theorem 19.** *The languages recognised by JMAs are those recognised by regular proofs.*

We prove the theorem in the next two subsections, by providing effective translations between the two models. Notice that by Rem. 17, the theorem implies that for regular proofs π, emptiness and other basic properties of $[\pi]$ are undecidable.

4.1 From JMAs to cyclic proofs

Let $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta\rangle$ be a jumping multihead automaton. We want to build a regular proof $\pi_{\mathcal{M}}$ of $A^* \vdash 2$ such that $[\pi_{\mathcal{M}}] = L(\mathcal{M})$. A difficulty is that heads in the automaton may stay in place, thus reading the same letter during several steps. In contrast the letters are read only once by cyclic proofs, so that we have to remember this information. We do so by labelling the sequents of the produced proof π_M with extra information describing the current state of the automaton. If $k' \in \mathbb{N}$, let $\mathcal{F}_{k'}$ be the set of injective functions $[\![1, k']\!] \to [\![1, k]\!]$. A *labelled sequent* is a sequent of the form $(A^*)^{k'} \vdash 2$ together with an extra label in $S \times \mathcal{F}_{k'} \times (A \cup \{\square, \triangleleft\})^k$.

The intuitive meaning of a label $(s, f, \vec{y})$ is the following: s is the current state of the automaton, f maps each formula A^* of the sequent to a head of the automaton, and $\vec{y}$ stores the letter that is currently processed by each head. Symbol $\square$ is used if this letter is unknown, and the head is scheduled to process this letter and move to the right. The values intuitively provided to each A^* formula of the sequent are the suffixes to the right of the corresponding heads of the automaton. On the examples, labels will be written in grey below the sequents.

It will always be the case that if the label of $(A^*)^{k'}$ is $(s, f, \vec{y})$, then $Im(f) \subseteq \{i \mid y_i \neq \triangleleft\}$, *i.e.* all heads reading symbols from $A \cup \{\square\}$ correspond to a formula A^* in the sequent. We say that a sequent is *fully labelled* if its label does not contain $\square$.

The construction of $\pi_{\mathcal{M}}$ will proceed by building gadgets in the form of proof trees, each one (apart from the initial gadget) connecting a labelled sequent in the conclusion to a finite set of labelled sequents in the hypotheses. If some labelled sequents in the hypotheses have already been encountered, we simply put back pointers to their previous occurrence. Since the number of labelled sequents is finite, this process eventually terminates and yields a description of $\pi_{\mathcal{M}}$.

When describing those gadgets we abbreviate sequences of inference steps or standalone proofs using double bars labelled with the involved rule names.

Initial gadget. The role of the initial gadget is to reach the first labelled sequent from the conclusion $A^* \vdash 2$. It simply creates k identical copies of A^*. This expresses the fact that the initial configuration is $\langle w, s_0, (0, 0, ...0)\rangle$. We note id_k the identity function on $[\![1, k]\!]$. The initial labelled sequent is $(A^*)^k \vdash 2$ together with label $(s_0, id_k, (\square, \dots, \square))$.

The initial gadget is as follows:

$$c, \dots, c\ \genfrac{}{}{1.5pt}{}{\underset{s_0, id_k, (\square, \dots, \square)}{(A^*)^k \vdash 2}}{A^* \vdash 2}$$

Reading gadget. Every time the label $(s, f, \vec{y})$ of the current address is not fully labelled, we use the gadget $read_i$, where $i = \min\{j \mid \vec{y}(j) = \square\}$ to process the first unknown letter.

We note $i' = f^{-1}(i)$ the position of the A^* formula corresponding to head i and define the gadget $read_i$ as follows:

$$\dfrac{\underset{s, f', (y_1, \dots, y_{i-1}, \triangleleft, \dots, y_k)}{(A^*)^{k'-1} \vdash 2} \qquad A\ \dfrac{\left(\underset{s, f, (y_1, \dots, y_{i-1}, a, \dots, y_k)}{(A^*)^{k'} \vdash 2}\right)_{a \in A}}{(A^*)^{i'-1}, A, A^*, (A^*)^{k'-i'} \vdash 2}}{\underset{s, f, (y_1, \dots, y_{i-1}, \square, \dots, y_k)}{(A^*)^{k'} \vdash 2}} *$$

where $f'(x) = \begin{cases} f(x) & \text{if } 1 \leq x < i' \\ f(x+1) & \text{if } i' \leq x \leq k' - 1 \end{cases}$

Transition gadget. Thanks to the $read_i$ gadgets, we can now assume we reach a fully labelled sequent, with label of the form $(s, f, (y_1, \dots, y_k))$. If $s \notin \{s_{acc}, s_{rej}\}$, we use a *transition gadget*, whose general shape is as on the right below, with $(s', \alpha) = \delta(s, (y_1, \dots, y_k))$:

This gadget is designed such that for all $i \in [\![1, k]\!]$:

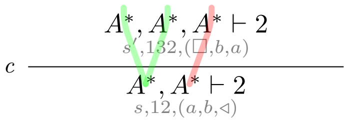
- if $\alpha(i) =$ ◌ then $z_i = y_i$
- if $\alpha(i) = \blacktriangleright\!\!|$ then $z_i = \square$,
- if $\alpha(i) = J_j$ then $z_i = y_j$.

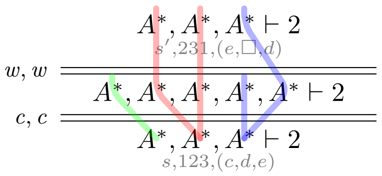
$$\delta\ \genfrac{}{}{1.5pt}{}{\underset{s', f', (z_1, \dots, z_k)}{(A^*)^{k''} \vdash 2}}{\underset{s, f, (y_1, \dots, y_k)}{(A^*)^{k'} \vdash 2}}$$

In the last case, a contraction is used to duplicate the A^* formula corresponding to head j, and the function f' maps this new formula to head i. The occurrence of A^* corresponding to y_i is weakened (possibly after having been duplicated if another head jumped to i).

We describe this gadget on two examples below. An element $f : [\![1, k']\!] \to [\![1, k]\!]$ is simply represented by $f(1)f(2)\dots f(k')$.

$\delta(s, (a, b, \lhd)) = (s', (\blacktriangleright\!\!|, \text{◌}, J_1))$ $\delta(s, (c, d, e)) = (s', (J_3, \blacktriangleright\!\!|, J_2))$





Notice that it is also possible to avoid unnecessary contractions, in order to bound the number of A^* formulas in a sequent by k. The symbol $\square$ means that the formula A^* is scheduled for a $*$ rule, and will be immediately processed thanks to the gadget $read_i$ as described above.

Final gadget. It remains to describe what happens if the current sequent is fully labelled with $s \in \{s_{acc}, s_{rej}\}$. In this case, we simply conclude with a $(\mathbf{tt})$ axiom if $s = s_{acc}$ or with a $(\mathbf{ff})$ axiom if $s = s_{rej}$.

This achieves the description of the preproof $\pi_{\mathcal{M}}$. The following lemma expresses its correctness; we prove it in [15, Appendix].

▶ **Lemma 20.** *If $\mathcal{M}$ is a progressing JMA, the preproof $\pi_{\mathcal{M}}$ is valid, and $[\pi_{\mathcal{M}}] = L(\mathcal{M})$.*

4.2 From cyclic proofs to JMAs

Let π be a regular proof with conclusion $A^* \vdash 2$. Let k be the maximal number of star formulas in the sequents of π. We build a JMA $\mathcal{M}$ with k heads such that $L(\mathcal{M}) = [\pi]$.

The idea of the construction is to store all necessary information on the current state of the computation in π into the state space of $\mathcal{M}$, besides the content of star formulas. This includes the current address in π, and the actual letters corresponding to the alphabet formulas, together with some information linking star formulas to heads of the automaton.

This allows $\mathcal{M}$ to mimic the computation of $[\pi]$ on an input u, in a similar way as the converse translation from Sect. 4.1. In particular, we keep the invariant that the value associated to each star formula is the suffix of u to the right of the corresponding head of $\mathcal{M}$.

State space of $\mathcal{M}$. Let m be the maximal number of alphabet formulas in the sequents of π. We use a register with m slots, each one possibly storing a letter from A. Let $R = \bigcup_{i=0}^{m} A^i$ be the set of possible register values. An element $b_1 \dots b_i$ of R describes the content of the i alphabet formulas of the current sequent. We denote the empty register by $\Diamond$. Intuitively, the register needs to store the values that have been processed by the automaton, but are still unknown in the proof π as they are represented by alphabet formulas.

Let $\mathcal{F}$ be the set $\bigcup_{i=0}^{k} [\![1, k]\!]^i$. An element $f \in \mathcal{F}$ associates to each A^* formula of a sequent the index of a head of $\mathcal{M}$. This allows us to keep track of the correspondence between heads of $\mathcal{M}$ and suffixes of the input word being processed by π.

We define the state space of $\mathcal{M}$ as $S = (Addr(\pi) \times R \times \mathcal{F}) \cup \{s_{acc}, s_{rej}\}$.

Notice that $Addr(\pi)$ is infinite, so $\mathcal{M}$ is an infinite-state JMA. However, if π has finitely many subtrees, we will be able to quotient $Addr(\pi)$ by $v \sim w$ if v and w correspond to the same subtree, and obtain a finite-state JMA.

If (v, r, f) is a state of $\mathcal{M}$, we will always have $|r| = m'$ and $|f| = k'$, where m' (resp. k') is the number of alphabet (resp. star) formulas in $\pi(v)$. Moreover, for all $i \in [\![1, m']\!]$, the i^{th} alphabet formula contains the letter $r(i)$ stored in the i^{th} slot of the register r.

The initial state is $s_0 = (\epsilon, \diamondsuit, 1)$. It points to the root of π, with empty register, and maps the only star formula to head 1.

Transition function of $\mathcal{M}$. If $s = (v, r, f)$ is a state of $\mathcal{M}$, and $\vec{a} = (a_1, \ldots, a_k)$ is the tuple of letters read by each head with $a_i \in A_{\triangleleft}$, we want to define $\delta(s, \vec{a}) = (s', \alpha) \in S \times Act^k$.

We write α_{id} for the action tuple (◌, . . . , ◌) leaving each head at the same position. We write $move_i$ (resp. $jump_{i,j}$) for the element of Act^k which associates to heads $i' \neq i$ the action ◌ and to head i the action ▶| (resp. jump to head j).

First of all, if the rule applied to v in π is an axiom ($\mathbf{tt}$) (resp. ($\mathbf{ff}$)), we set $s' = s_{acc}$ (resp. s_{rej}) and $\alpha = \alpha_{id}$. This allows $\mathcal{M}$ to stop the computation and return the same value as $[\pi]$. Otherwise, we define $s' = (v', r', f')$ and α depending on the rule applied to v in π. By Prop. 10, we can assume that the proof π does not use the weakening rule. Let m' (resp. k') be the number of alphabet (resp. star) formulas in $\pi(v)$.

Contraction Rule: We set $v' = v0$, and do a case analysis on the principal formula:

- i^{th} alphabet formula: we set $f' = f$, $r' = r(1) \cdots r(i-1) \cdot r(i) \cdot r(i) \cdot r(i+1) \cdots r(m')$ and $\alpha = \alpha_{id}$.
- i^{th} star formula: let $j \in [\![1, k]\!]$ be the smallest integer not appearing in f, corresponding to the index of the first available head. We want to allocate it to this new copy, by making it jump to the position of the head $f(i)$. We take $r' = r$, $f' = f(1) \cdots f(i) \cdot j \cdot f(i+1) \cdots f(k')$, and $\alpha = jump_{j,f(i)}$.

Star rule: Let i be the index of the principal star formula. We now want the head $j \triangleq f(i)$ pointing on this formula to move right. The letter processed by this head will be added to the register.

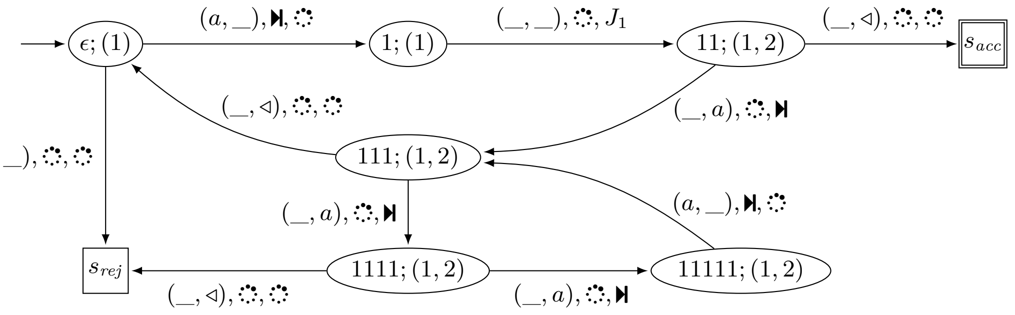
- if $\vec{a}(j) = \triangleleft$, the head reached the end of the input. This corresponds to the left premiss of the $*$ rule. We set $v = v0$, $f' = f(1) \cdots f(i-1) f(i+1) \cdots f(k')$, $r' = r$ and $\alpha = \alpha_{id}$.
- if $\vec{a}(j) \in A$, we set $v' = v1$, $f' = f$, $\alpha = move_i$, and $r' = r(1) \cdots r(i') \vec{a}(i) r$ $(i'+1) \cdots r(m')$, where i' is the number of A formulas before the principal star formula.

Let i be the index of the principal A formula, and $a = r(i)$ be the letter associated to it. We define $v' = va$, $f' = f$, $\alpha = \alpha_{id}$, and $r' = r(1) \ldots r(i-1) r$ $(i+1) \ldots r(m')$, *i.e.* we erase the i^{th} slot.

This completes the description of the JMA $\mathcal{M} = \langle S, k, s_0, s_{acc}, s_{rej}, \delta \rangle$.

▶ **Lemma 21.** *The JMA $\mathcal{M}$ is progressing, and $L(\mathcal{M}) = [\pi]$.*

▶ **Example 22.** We can obtain a progressing JMA for the language $L = \{a^{2^n} \mid n \in \mathbb{N}\}$ by translating the proof from Fig. 4 using the above procedure. As there are at most two star formulas in the sequents of the proof, the produced JMA has two heads. As there is only one letter in the alphabet, we can just forget the register. Similarly we consider that any $\mathbf{ff}$ part (resp. $\mathbf{tt}$ part) of the proof corresponds to the state s_{rej} (resp. s_{acc}). Using _ for reading any symbol (a letter a or $\triangleleft$), we can represent the obtained automaton as follows:



▶ Remark 23. Our encoding from regular proofs to JMAs would still work if we had included an exchange rule in the system, and the encoding from JMAs to regular proofs does not require the exchange rule. Therefore, such a rule would not increase the expressive power.

4.3 The affine case: regular languages

Looking at the encodings in the two previous sections, we can observe that:

- the encoding of an affine regular proof is a JMA with a single head: in absence of contraction, all sequents in proof ending with $A^* \vdash 2$ have at most one star formula;
- the encoding of a JMA with a single head does not require contraction: this rule is used only for the initial gadget and when the action of a head is to jump on another one.

As a consequence, we have a correspondence between affine regular proofs and JMAs with a single head, whence, by Lemma 16:

▶ **Theorem 24.** *The regular languages are those recognisable by affine regular proofs.*

5 Conclusion

We have defined a cyclic proof system where proofs denote formal languages, as well as a new automata model: jumping multihead automata. We have shown that regular proofs correspond precisely to the languages recognisable by jumping multihead automata, and that affine regular proofs correspond to regular languages. We see two directions for future work.

First, we restricted to sequents of the shape $E \vdash 2$ in order to focus on languages. The proof system we started from (LKA [7]) however makes it possible to deal with sequents of the shape $E \vdash e$: it suffices to include right introduction rules for the alphabet (A) and star formulas (A^*). By doing so, we obtain a system where proofs of $A^* \vdash A^*$ denote *transductions*: functions from words to words. We conjecture that in the affine case, we obtain exactly the *subsequential transductions* [17], *i.e.* transductions definable by deterministic 1-way transducers. In the general case (with contraction), we would need a notion of jumping multihead transducers.

Second, we used a *cut-free* proof system. While adding the cut rule for the presented system (restricted to sequents $E \vdash 2$) seems peculiar since the input and output are not of the same shape, it becomes reasonable when moving to general sequents for transductions. We have observed that we can go beyond MSO-definable transductions when doing so, even in the affine case. We would like to investigate and hopefully characterise the corresponding class of transductions.

References

1 Bahareh Afshari and Graham E. Leigh. Cut-free completeness for modal mu-calculus. In *LiCS*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005088.

2 Stefano Berardi and Makoto Tatsuta. Classical System of Martin-Löf's Inductive Definitions Is Not Equivalent to Cyclic Proof System. In *FoSSaCS*, pages 301–317, 2017. doi:10.1007/978-3-662-54458-7_18.

3 Stefano Berardi and Makoto Tatsuta. Equivalence of inductive definitions and cyclic proofs under arithmetic. In *LiCS*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005114.

4 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2011. doi:10.1093/logcom/exq052.

5 Anupam Das, Amina Doumane, and Damien Pous. Left-handed completeness for Kleene algebra, via cyclic proofs. In *LPAR*, volume 57 of *EPiC Series in Computing*, pages 271–289. Easychair, 2018. doi:10.29007/hzq3.

6 Anupam Das and Damien Pous. A cut-free cyclic proof system for Kleene algebra. In *TABLEAUX*, volume 10501 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2017. doi:10.1007/978-3-319-66902-1_16.

7 Anupam Das and Damien Pous. Non-wellfounded proof theory for (Kleene+action)(algebras+lattices). In *CSL*, volume 119 of *LIPIcs*, pages 18:1–18:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CSL.2018.19.

8 Christian Dax, Martin Hofmann, and Martin Lange. A Proof System for the Linear Time μ-Calculus. In *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006. doi:10.1007/11944836_26.

9 Amina Doumane. Constructive completeness for the linear-time μ-calculus. In *LiCS*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005075.

10 Amina Doumane, David Baelde, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. In *CSL*, volume 62 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, September 2016. doi:10.4230/LIPIcs.CSL.2016.42.

11 Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *CSL*, volume 23 of *LIPIcs*, pages 248–262, 2013. doi:10.4230/LIPIcs.CSL.2013.248.

12 Alain Frisch and Luca Cardelli. Greedy Regular Expression Matching. In *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 618–629. Springer, 2004. doi:10.1007/978-3-540-27836-8_53.

13 Fritz Henglein and Lasse Nielsen. Regular expression containment: coinductive axiomatization and computational interpretation. In *POPL, 2011*, pages 385–398. ACM, 2011. doi:10.1145/1926385.1926429.

14 Markus Holzer, Martin Kutrib, and Andreas Malcher. Multi-Head Finite Automata: Characterizations, Concepts and Open Problems. In *International Workshop on The Complexity of Simple Programs (CSP)*, pages 93–107, 2008. doi:10.4204/EPTCS.1.9.

15 Denis Kuperberg, Laureline Pinault, and Damien Pous. Cyclic Proofs and Jumping Automata, 2019. Version with appendix. URL: https://hal.archives-ouvertes.fr/hal-02301651.

16 Alexander Meduna and Petr Zemek. Jumping Finite Automata. *Int. J. Found. Comput. Sci.*, 23(7):1555–1578, 2012. doi:10.1142/S0129054112500244.

17 Marcel Paul Schützenberger. Sur une Variante des Fonctions Sequentielles. *Theor. Comput. Sci.*, 4(1):47–57, 1977.

18 Alex Simpson. Cyclic Arithmetic Is Equivalent to Peano Arithmetic. In *FoSSaCS*, pages 283–300, 2017. doi:10.1007/978-3-662-54458-7_17.

19 Andrew C Yao and Ronald L. Rivest. k+1 Heads Are Better than k. *Journal of the ACM*, 25(2):337–340, 1978. doi:10.1145/322063.322076.

Reachability in Concurrent Uninterpreted Programs

Salvatore La Torre
Università degli Studi di Salerno, Italia
`https://docenti.unisa.it/salvatore.latorre`
slatorre@unisa.it

Madhusudan Parthasarathy
University of Illinois, Urbana-Champaign, USA
`http://madhu.cs.illinois.edu/`
madhu@illinois.edu

Abstract

We study the safety verification (reachability problem) for concurrent programs with uninterpreted functions/relations. By extending the notion of coherence, recently identified for sequential programs, to concurrent programs, we show that reachability in coherent concurrent programs under various scheduling restrictions is decidable by a reduction to multistack pushdown automata, and establish precise complexity bounds for them. We also prove that the coherence restriction for these various scheduling restrictions is itself a decidable property.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Formal languages and automata theory; Software and its engineering → Formal software verification

Keywords and phrases Verification, uninterpreted programs, concurrent programs, shared memory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.46

Acknowledgements This material is based upon work supported by the National Science Foundation under Grant CCF-1527395, GNCS 2019 grant and MIUR-FARB 2018-19 grant.

1 Introduction

Verification against assertion violations for sequential programs that have only Boolean variables and have recursive function calls is decidable, as the problem is equivalent to pushdown automata reachability/emptiness. However, generalizations of this result to other settings is hard. First, there are hardly any positive results for verifying recursive programs that work over infinite domains. Second, concurrent recursive program verification is typically undecidable (two stacks suffice to encode the executions of Turing machines) if the interaction of the threads is not restricted in any way.

A recent paper by Mathur et al. introduces a decidable class of sequential programs where the data domain is infinite [27]. The first ingredient for decidability is that the programs compute terms over functions and compare them over relations that are both assumed to be *uninterpreted*. The theory of uninterpreted functions is an important theory. Theoretically, it was the one studied by Gödel for his completeness theorems [15], and practically, the decidability of validity of its quantifier-free fragment is exploited by SMT solvers and is often used (typically in combination with other theories) to solve feasibility of loop-free program snippets, in bounded model-checking, and to validate verification conditions [10]. The second ingredient for decidability is a technical restriction called *coherence*. Coherent programs have two properties – the *memoizing property* (which intuitively says that computed terms once dropped cannot be recomputed) and the *early assume property* (which intuitively says that equality assumptions in program executions happen early, well before their superterms

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 46; pp. 46:1–46:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are computed and dropped). The work by [27] shows that for coherent programs, one can build a streaming congruence closure algorithm with finite memory, and by modeling this algorithm as an automaton, verify programs.

From a practical point of view, uninterpreted abstractions have been considered [18], and a recent paper shows that verifying programs using an uninterpreted abstraction can be effective [12]. Also, extensions of the decidable verification result for uninterpreted programs has found applications in verifying memory-safety for heap-manipulating programs, where heaps are naturally modeled using infinite domains [28].

In this paper, we consider the problem of verifying concurrent uninterpreted programs, with both recursion and shared memory.

Note that concurrent recursive programs even over Boolean domains have an undecidability verification (reachability) problem. Programs with uninterpreted functions/relations are much more complex than programs with Boolean domains – it is easy to see that we can simulate a program with Boolean domains by using a program with no functions or relations, with two special immutable variables for T and F, and only using equality relations in the program. Moreover, this will always yield a coherent program. Consequently, concurrent recursive coherent uninterpreted programs clearly have an undecidable reachability problem.

There has been a rich literature of work that has identified restrictions of concurrent recursive Boolean programs for which reachability is decidable (see [2, 9, 19, 21, 24, 25, 30]). Verification of such programs can be modeled as reachability/emptiness problem in *multistack pushdown systems* and several underapproximations based on restricting the scheduling of threads has yielded decidability. These include bounded context-switching [30], bounded scope executions [24], (k, d)-budgeted executions [2], k-phase executions [20], k-path-tree executions [23], etc. Some generalizations of the above decidability results that show decidability when manipulating multiple stacks in a way that the accesses correspond to bounded tree-width manipulations are also known [13, 26], and some of these restrictions have been applied for finding errors in predicate abstracted programs as well [17, 31, 21]. There has also been a lot of work on studying register automata (both sequential and concurrent) on *data words* where registers can read data from infinite domains but use only equality/disequality checks (note that uninterpreted functions/relations that must satisfy the congruence axioms are not allowed) [6, 7, 14, 8, 29]; see also work extending to programs where variables range over natural numbers, with equality, but with no functions [1].

The goal of this paper is to establish decidability results for concurrent recursive programs working over *infinite* uninterpreted data domains with certain scheduling restrictions that were shown to yield decidable reachability in the setting of Boolean programs. Our main results are that coherent concurrent program safety verification (for a notion of coherence for concurrent programs we define) is decidable for bounded context-switching, bounded scope executions, (k, d)-budgeted executions, k-phase executions, ordered executions, and k-path-tree executions.

There are two primary technical challenges to establish our results. First, we need to define an appropriate extension of coherence for concurrent programs. We propose such a natural extension. We model a concurrent program as working on a data-domain that is shared amongst all processes; we think this is an important design decision as the alternative choice of having local universes is both unnatural (domains of programs in the real world are often common, like integers or other forms of data structures) and prohibits communication of unbounded data between processes. The notion of coherence (memoizing and early assumes) is defined based on the *frontier* of computation, which includes all variables that could ever come into scope. This includes the unbounded copies of local variables stored in each program's stack as well as the shared variables.

The second challenge is to build a multistack automaton that accurately captures the feasibility of coherent runs of the concurrent recursive program. In the automata constructed in [27] for sequential programs, the automata do not store actual elements of the universe in the stack or state, of course, as the universe is infinite. Rather, the automata store *relationships* between variables – more precisely the equalities between variables in the *initial model* defined by the equality assumptions occurring in the program's execution, the disequality constraints implied by it, and certain local function maps between variables. Let us call this the *EDF information* – information on equalities, disequalities, and function maps between variables. The automaton constructed by [27] effects a *streaming congruence closure* algorithm that has finite memory by keeping track of the EDF information on variables currently in scope after any execution.

In concurrent programs, the EDF information is significantly harder to keep track of because it includes relationships between not just the local variables of one process or relationships between local variables and shared variables, but also between local variables of *different* processes. For example, when a call returns in a local thread, it has to recover all EDF information between new local variables in scope and local variables of other threads.

The above complications mean that we *cannot* simply abstract each local thread into its EDF information and then take their concurrent evolution. In fact, our results do *not* extend to parameterized concurrent systems (where there are an unbounded number of processes), even for the cases where Boolean program verification under certain scheduling restrictions are known to be decidable (e.g., *bounded rounds* is known to be decidable [21]).

Our multistack automaton construction instead maintains a complex invariant – an element in the stack for a process p contains several kinds of information: the EDF information on local variables of p and shared variables at the time the push (function call) happened, EDF relationship of local variables in p to local variables in p just below the stack (the caller's variables), and most importantly, EDF information between local variables across processes *at the time this information was pushed onto the stack.* Maintaining this complex invariant at every stage is involved, and gives us the reduction from reachability of concurrent coherent programs to multistack automata reachability.

The reduction to multistack automata reachability is rewarding as we can exploit the fact that reachability of the latter under various scheduling restrictions have been well studied for establishing decidability. Utilizing these results, we show that concurrent coherent program reachability is decidable for the following restrictions: bounded context-switching, bounded scope executions (k, d)-budgeted executions, k-phase executions, ordered executions, and k-path-tree executions. We also show that our decidability results have optimal complexity. In fact we can show that the complexity of verification of coherent concurrent programs is precisely the same as that for Boolean programs under similar scheduling restrictions.

There is another natural related question that arises: Given a concurrent program with a scheduling restriction as above, how can the user determine whether it is coherent? We show that checking whether a concurrent program is coherent under these scheduling restrictions is also decidable and decidable in the same time complexity as the reachability algorithm.

2 Concurrent Uninterpreted Programs

In this section, we introduce the notion of concurrent programs over data domains with uninterpreted functions and relations. We start by recalling some definitions about first order data structures, then we recall the definition of uninterpreted sequential programs and then we extend it to concurrent programs.

A first order *signature* is $(\mathcal{C}, \mathcal{F}, \mathcal{R})$ where $\mathcal{C}$ is a set of constants, $\mathcal{F}$ is a set of function symbols, and $\mathcal{R}$ is a set of relation symbols. Relations and functions have an implicitly assigned arity in $\mathbb{N}_{>0}$. A signature is *algebraic* if $\mathcal{R}$ is empty and in this case we denote it simply as the pair $(\mathcal{C}, \mathcal{F})$. A *data model* for $(\mathcal{C}, \mathcal{F}, \mathcal{R})$ is $\mathcal{M} = (U, \{[\![c]\!] \mid c \in \mathcal{C}\}, \{[\![f]\!] \mid f \in \mathcal{F}\}, \{[\![R]\!] \mid R \in \mathcal{R}\})$ consisting of a universe, and an interpretation on the universe for constants, functions and relations (a data model for an algebraic signature will not have an interpretation for the relations). The set of *terms* is defined inductively as follows: each constant from $\mathcal{C}$ is a term and for any m-ary function f and terms $t_1, \ldots, t_m$, $f(t_1, \ldots, t_m)$ is also a term. An *immediate superterm* of t is $f(t_1, \ldots, t_m)$ where $t \in \{t_1, \ldots, t_m\}$. A *superterm* of t is either an immediate superterm of t or an immediate superterm of a superterm of t. The interpretation of a term t in $\mathcal{M}$ is denoted as $[\![t]\!]_{\mathcal{M}}$.

In the following, for an integer $n > 0$, we denote with $[n]$ the set $\{1, \ldots, n\}$.

2.1 Uninterpreted sequential programs

We consider simple *sequential* programs over uninterpreted functions and relations, and with possibly recursive calls to methods. We fix a finite set of variables V which includes both local and global variables used by the programs to store information during a computation. Values are manipulated by using function and relation symbols from a first order signature $(\mathcal{C}, \mathcal{F}, \mathcal{R})$. We also fix a finite set of method names M. A program is essentially formed of a list of method definitions, one for each method name in M and such that there is a *main* method, i.e., the method from which the execution starts, that we denote m_0. We allow methods to return tuples of values, thus for every method $m \in M$, we fix a tuple of distinct *output* variables $\mathtt{o}_m$. Also for the ease of presentation and without loss of generality, we assume that all methods have the same list of parameters which coincides with a fixed permutation of all the local variables. In the following, we denote such list as *lvars*. Each method body contains assignments, sequencing, conditionals, loops and method calls.

The precise syntax is given by the following grammar:

$$\begin{array}{lll}
\langle pgm \rangle & ::= & m \Rightarrow \mathtt{o}_m \langle\, stmt \rangle \mid \langle pgm \rangle\ \langle pgm \rangle \\
\langle stmt \rangle & ::= & \langle stmt \rangle \mathtt{;}\ \langle stmt \rangle \mid \mathtt{skip} \mid x := y \mid x := f(\mathtt{z}) \mid \mathtt{assume}(\langle cond \rangle) \\
 & & \mid \mathtt{w} := m(lvars) \mid \mathtt{if}\ (\langle cond \rangle)\ \mathtt{then}\ \langle stmt \rangle\ \mathtt{else}\ \langle stmt \rangle \mid \mathtt{while}\ (\langle cond \rangle)\ \langle stmt \rangle \\
\langle cond \rangle & ::= & x = y \mid x = c \mid c = d \mid R(\mathtt{z}) \mid \langle cond \rangle \vee \langle cond \rangle \mid \neg \langle cond \rangle
\end{array}$$

In the above, $m \in M$ is a method name, $c, d \in \mathcal{C}$ are constants, $f \in \mathcal{F}$ is a function name, $R \in \mathcal{R}$ is a relation name, $x, y \in V$ are variables, $\mathtt{w}$ is a tuple of variables from V, and $\mathtt{z}$ is a tuple of constants from $\mathcal{C}$ and variables from V. Moreover, we allow for standard operators: ':=' is the assignment operator, ';' is the program sequencing operator, `skip` is the "do nothing" statement, `if-then-else` is the usual conditional statement and `while` is the usual loop statement. Method calls are handled as usual with a call stack. Namely, a *configuration* of the program consists of a *stack* which stores the history of positions at which calls were made, along with valuations for local variables, and the top of the stack contains the local and global valuations, and a pointer to the current statement being executed. Note that we do not make use of an explicit `return` statement: a call to module m is returned when there are no more statements of m to execute (the values that need to be returned are assigned to the output variables $\mathtt{o}_m$ before the call ends).

For the ease of presentation, in the rest of the paper we will assume that the programs have only conditionals of the form '$x = y$' and '$x \neq y$', constants will not appear in any of the program expressions and the signature is algebraic, i.e., $\mathcal{R}$ is empty. Note that this is without loss of generality. In fact, any relation R can be captured by a function f_R with the same arity and a Boolean variable b_R. A Boolean combination of conditions can be modeled using the `if-then-else` construct. Constants can be removed by using instead variables that are not modified in the program.

Fix a set of functions $\mathcal{F}$. An *execution* of a sequential program over a set of variables V and set of methods M is a sequence over the alphabet $\Pi = \{``x := y", ``x := f(\mathtt{z})", ``\mathtt{assume}(x = y)", ``\mathtt{assume}(x \neq y)", ``\mathtt{call}\ m", ``\mathtt{w} := \mathtt{return}" \mid x, y, \mathtt{z}, \mathtt{w} \text{ are in } V, m \in M\}$. In particular, for a program P, denoting $bd(m)$ the body of a method $m \in M$, the set of *complete executions* of P is generated by the following context-free grammar:

$$\begin{array}{rcl}
X_\varepsilon & \rightarrow & \varepsilon \\
X_{\mathtt{skip};\, st} & \rightarrow & X_{st} \\
X_{x:=y;\, st} & \rightarrow & ``x := y" \cdot X_{st} \\
X_{x:=f(\mathtt{z});\, st} & \rightarrow & ``x := f(\mathtt{z})" \cdot X_{st} \\
X_{\mathtt{assume}(c);\, st} & \rightarrow & ``\mathtt{assume}(c)" \cdot X_{st} \\
X_{\mathtt{if}(c)\ \mathtt{then}\ st_1\ \mathtt{else}\ st_2\ ;\, st} & \rightarrow & ``\mathtt{assume}(c)" \cdot X_{st_1\, ;\, st} \mid ``\mathtt{assume}(\neg c)" \cdot X_{st_2\, ;\, st} \\
X_{\mathtt{while}(c)\ \{st_1\}\ ;\, st} & \rightarrow & ``\mathtt{assume}(c)" \cdot X_{st_1\, ;\, \mathtt{while}(c)\ \{st_1\}\ ;\, st} \mid ``\mathtt{assume}(\neg c)" \cdot X_{st} \\
X_{\mathtt{w}:=m(lvars)}\ ;\ st & \rightarrow & ``\mathtt{call}\ m" \cdot X_{bd(m)} \cdot ``\mathtt{w} := \mathtt{return}" \cdot X_{st}
\end{array}$$

where X_{st} denotes the nonterminal symbol corresponding to a statement st and $X_{bd(m_0)\, ;\, \varepsilon}$ is the start symbol (recall m_0 is the main method). An *execution* is any prefix of a complete execution. Note that not all the executions are feasible.

2.2 Concurrent uninterpreted programs

A *concurrent uninterpreted program* is a finite set of recursive uninterpreted programs running in parallel and sharing a finite set of variables S. The syntax of concurrent programs is defined by extending the syntax of the sequential programs with the following rule:

$$\langle \textit{conc-pgm} \rangle \ ::= \ \langle \textit{pgm} \rangle \ \mid \ \langle \textit{pgm} \rangle \parallel \langle \textit{conc-pgm} \rangle$$

where each sequential program uses its own set of local and global variables along with the set of the shared variables (for a sequential program the shared variables are as global variables, the only difference is that they can be read and written also by the other sequential threads).

For the rest of the paper, we fix a concurrent uninterpreted program $\mathcal{P}$ that is formed by the sequential programs $P_1, \ldots, P_n$ (where $n > 0$). We refer to programs $P_1, \ldots, P_n$ as the *component programs* of $\mathcal{P}$. We denote with V_i the set of local and global variables of each P_i and assume that the sets $S, V_1, \ldots, V_n$ are pairwise disjoint. Further, we denote $\mathtt{Vars} = S \cup \bigcup_{i=1}^{n} V_i$ the set of all variables used in $\mathcal{P}$. Any (*complete*) *execution* of $\mathcal{P}$ is obtained as an interleaving $\rho_1, \ldots, \rho_n$ where ρ_i is a (complete) execution of P_i for $i \in [n]$.

To distinguish among the different sequential programs in a concurrent execution, we assume pairwise disjoint alphabets and for each program P_i, we will denote the corresponding alphabet Π_i and any symbol of the form "a" as $\langle a \rangle_i$, that is, we let $\Pi_i = \{\langle x := y \rangle_i, \langle x := f(\mathtt{z}) \rangle_i, \langle \mathtt{assume}(x = y) \rangle_i, \langle \mathtt{assume}(x \neq y) \rangle_i, \langle \mathtt{call}\ m \rangle_i, \langle \mathtt{w} := \mathtt{return} \rangle_i \mid x, y, \mathtt{z}, \mathtt{w} \text{ are in } V_i \cup S, m \in M\}$. Thus, the overall alphabet for $\mathcal{P}$ is $\Pi = \bigcup_{i \in [n]} \Pi_i$.

At the beginning of any execution each variable x is set according to an initial interpretation of the data model that we denote with $\mathtt{init}(x)$. Then, the variables are updated according to the intended semantics of the statements.

For any ρ' such that $\rho \cdot \rho' \cdot \rho''$ is an execution of $\mathcal{P}$ and $i \in [n]$, we say that ρ' is *i-matched* if all the calls of program P_i (i.e., of the form $\langle \mathtt{call}\ m \rangle_i$) that occur in ρ' are matched within ρ'. The map `Comp` captures the term associated with program variables at the end of any execution. Denoting with ρ any execution of $\mathcal{P}$, `Comp` is inductively defined as follows:

$$\begin{array}{rcll}
comp(\varepsilon, x) & = & \mathtt{init}(x) & x \in \mathtt{Vars} \\
\mathtt{Comp}(\rho.\langle x := y\rangle_i, x) & = & \mathtt{Comp}(\rho, y) & \\
\mathtt{Comp}(\rho.\langle x := y\rangle_i, x') & = & \mathtt{Comp}(\rho, x') & x' \in \mathtt{Vars} \text{ and } x' \neq x \\
\mathtt{Comp}(\rho \cdot \langle x := f(\mathtt{z})\rangle_i, x) & = & f(\mathtt{Comp}(\rho, z_1), \ldots, \mathtt{Comp}(\rho, z_r)) & \text{where } \mathtt{z} = (z_1, \ldots, z_r) \\
\mathtt{Comp}(\rho \cdot \langle x := f(\mathtt{z})\rangle_i, x') & = & \mathtt{Comp}(\rho, x') & x \neq x' \\
\mathtt{Comp}(\rho \cdot \langle \mathtt{assume}(x = y)\rangle_i, x') & = & \mathtt{Comp}(\rho, x') & x' \in \mathtt{Vars} \\
\mathtt{Comp}(\rho \cdot \langle \mathtt{assume}(x \neq y)\rangle_i, x') & = & \mathtt{Comp}(\rho, x') & x' \in \mathtt{Vars} \\
\begin{array}{r}\mathtt{Comp}(\rho \cdot \langle \mathtt{call}\ m\rangle_i \cdot \rho' \\ \cdot\langle (w_1, \ldots w_r := \mathtt{return}\rangle_i, w_j)\end{array} & = & \mathtt{Comp}(\rho \cdot \langle \mathtt{call}\ m\rangle_i \cdot \rho', \mathtt{o}_m[j]) & \rho' \text{ is } i\text{-matched} \\
\begin{array}{r}\mathtt{Comp}(\rho \cdot \langle \mathtt{call}\ m\rangle_i \cdot \rho' \\ \cdot\langle (w_1, \ldots w_r := \mathtt{return}\rangle_i, x)\end{array} & = & \mathtt{Comp}(\rho, x) & \begin{array}{l}\rho' \text{ is } i\text{-matched,} \\ x \notin \{w_1, \ldots w_r\}\end{array}
\end{array}$$

We denote with $\leq$ the prefix relation among executions, i.e., for executions ρ, ρ', with $\rho' \leq \rho$ we mean that ρ' is a prefix of ρ. The set of all the terms computed by an execution ρ is $\mathtt{Terms}(\rho) = \bigcup_{\rho' \leq \rho, x \in \mathtt{Vars}} \mathtt{Comp}(\rho', x)$.

The semantics of uninterpreted programs is defined with respect to a data model that gives a meaning to the elements in the signature, and thus to the computed terms. An execution ρ is said to be *feasible* with respect to a data model if the assumptions it makes are true in that model. To formalize the notion of feasible executions we first define the sets of the equality assumes and the disequality assumes of an execution.

For any execution ρ, the set of *equality assumes* of ρ, denoted $\alpha(\rho)$, is a subset of $\mathtt{Terms}(\rho) \times \mathtt{Terms}(\rho)$ inductively defined as: $\alpha(\varepsilon) = \emptyset$; and if σ is $\langle \mathtt{assume}(x = y)\rangle_i$, $i \in [n]$, then $\alpha(\rho \cdot \sigma) = \alpha(\rho) \cup \{(\mathtt{Comp}(\rho, x), \mathtt{Comp}(\rho, y))\}$, otherwise $\alpha(\rho \cdot \sigma) = \alpha(\rho)$. Similarly, the set of *disequality assumes* $\beta(\rho)$ can be defined as: $\beta(\varepsilon) = \emptyset$; and if σ is $\langle \mathtt{assume}(x \neq y)\rangle_i$, $i \in [n]$, then $\beta(\rho \cdot \sigma) = \beta(\rho) \cup \{(\mathtt{Comp}(\rho, x), \mathtt{Comp}(\rho, y))\}$, otherwise $\beta(\rho \cdot \sigma) = \beta(\rho)$.

An execution ρ is *feasible* in a data-model $\mathcal{M}$ if $[\![t]\!]_{\mathcal{M}} = [\![t']\!]_{\mathcal{M}}$ for every $(t, t') \in \alpha(\rho)$, and $[\![t]\!]_{\mathcal{M}} \neq [\![t']\!]_{\mathcal{M}}$ for every $(t, t') \in \beta(\rho)$.

We recall that an equivalence relation $\cong \subseteq \mathtt{Terms} \times \mathtt{Terms}$ is said to be a *congruence* if whenever $t_1 \cong t'_1$, $t_2 \cong t'_2$, $\ldots t_m \cong t'_m$ and f is an m-ary function then $f(t_1, \ldots t_m) \cong f(t'_1, \ldots t'_m)$. Given a binary relation $A \subseteq \mathtt{Terms} \times \mathtt{Terms}$, the *congruence closure* of A, denoted $\cong_A$, is the smallest congruence containing A. We can then show:

▶ **Proposition 1.** *An execution ρ is feasible in some data model if and only if $\cong_{\alpha(\rho)} \cap \beta(\rho) = \emptyset$.*

3 Verification of concurrent uninterpreted programs

The basic verification problem is *reachability* that consists of checking whether a given set of target states is reachable in a program execution. For uninterpreted programs, there is an additional request: the execution must be feasible in some data model. The target set is often captured by a program counter (corresponding to an assertion) and a Boolean combination of equalities over program variables. As also observed in [27], by simple program transformations, the reachability problem for uninterpreted programs can always be reduced to checking the existence of a feasible complete execution (the assertion condition is translated into a block containing `assume` and `if` statements). In such a translation the size of the resulting program is linear in the sizes of the starting program and the assertion conditions. Thus we consider the following reachability problem for concurrent uninterpreted programs:

▶ **Definition 2** (Reachability). *Given a concurrent uninterpreted program $\mathcal{P}$, the reachability problem asks whether there exists a feasible complete execution of $\mathcal{P}$.*

We observe that this decision problem is already undecidable for sequential uninterpreted programs even in absence of recursive method calls, and becomes EXPTIME-complete if the search is restricted to coherent computations [27]. Unfortunately, in the case of concurrent uninterpreted programs, assuming coherence does not suffice to gain decidability. In fact, reachability is undecidable even for concurrent programs with variables ranging over finite domains and with only two component programs. Consequently, we further restrict the executions by adopting some limitations studied in the literature that bound the interaction among the component programs.

In the rest of this section, we define first the notion of coherence, then the above mentioned restrictions for concurrent uninterpreted programs, and then the general bounded reachability problem. Finally, we conclude with a high level description of our approach to decide it.

Coherence. For terms $t_1, t_2 \in \texttt{Terms}$ and congruence $\cong$ on $\texttt{Terms}$, we say that t_2 *is a superterm of* t_1 *modulo* $\cong$ if there are terms $t'_1, t'_2 \in \texttt{Terms}$ such that $t'_1 \cong t_1$, $t'_2 \cong t_2$ and t'_2 is a superterm of t'_1. For the ease of presentation in the informal descriptions we will identify equivalent terms, and thus we will refer to a term meaning any of its equivalent terms. For example, if we say that a term t is recomputed in an execution, we actually mean that the computed term t is equivalent to a term that was computed earlier in the execution. Also, we call a superterm modulo an equivalence $\cong$ simply a superterm.

The notion of coherent execution introduced in [27] for sequential programs naturally extends to concurrent programs by using the $\texttt{Comp}$ map defined above. Informally, an execution is coherent if it is memoizing and has early assumes. The memoizing property says that if a term t is recomputed there must be a variable that currently evaluates to t. The early assume property instead imposes constraints on when $\langle\texttt{assume}(x = y)\rangle_i$ steps are taken within the execution: it requires that such $\texttt{assume}$ statements appear before the execution reassigns all the variables storing any computed term t that is a superterm of the terms stored in x or y (i.e., before it "drops" all of such superterms).

Formally, we say that a (complete) execution ρ over variables $\texttt{Vars}$ is *coherent* if it satisfies the following two properties:

1. (*Memoizing*) Let $\pi' = \pi \cdot \langle x := f(\texttt{z})\rangle_i$ be a prefix of ρ and let $t = \texttt{Comp}(\pi', x)$. If there is a term $t' \in \texttt{Terms}(\pi)$ such that $t' \cong_{\alpha(\pi)} t$, then there must exist some $y \in V$ such that $\texttt{Comp}(\pi, y) \cong_{\alpha(\pi)} t$.
2. (*Early Assumes*) Let $\pi' = \pi \cdot \langle\texttt{assume}(x{=}y)\rangle$ be a prefix of ρ and let $t_x = \texttt{Comp}(\pi, x)$ and $t_y = \texttt{Comp}(\pi, y)$. If there is a term $t' \in \texttt{Terms}(\pi)$ such that t' is either a superterm of t_x or of t_y modulo $\cong_{\alpha(\pi)}$, then there must exist a variable $z \in V$ such that $\texttt{Comp}(\pi, z) \cong_{\alpha(\pi)} t'$.

A *coherent program* is a program whose executions are all coherent.

In the literature, there is a notion of *freshness* [32, 8] that may remind people of the notion of memoizing above; however, these are not similar, as the memoizing restriction is on freshness of the computed *terms* and not on the underlying semantics of the data values computed (two terms may be different but still correspond to the same element in a particular data model).

Bounding the interaction among the component programs. Denote with $\Pi = \bigcup_{i\in[n]} \Pi_i$ the alphabet over variables $\texttt{V}$ and functions $\mathcal{F}$. Also, for any ρ' such that $\rho \cdot \rho' \cdot \rho''$ is an execution of $\mathcal{P}$ and $i \in [n]$, we say that ρ' is *i-matched* if all the calls of program P_i (i.e., of the form $\langle\texttt{call}\ m\rangle_i$) that occur in ρ' are matched within ρ'. For integers $k, d > 0$, we consider the *bounding conditions* that restrict the search respectively to the following sets of executions:

- a *k-context execution* ρ is the concatenation of k contexts, i.e., $\rho = \rho_1 \dots \rho_k$ where $\rho_i \in \Pi_{j_i}^*$, for $i \in [k]$ and $j_i \in [n]$, is a *context* of P_{j_i} (*bounded context-switching*, CON for short) [30];
- a *k-scoped execution* ρ is such that for each pair of matching call and return from any Π_i, the portion of ρ delimited by them does not contain more than k contexts of P_i, i.e., for any decomposition $\rho = \rho'.\sigma_c.\rho''.\sigma_r.\rho'''$ where for some $i \in [n]$, σ_c is of the form $\langle \mathtt{call}\ m \rangle_i$, σ_r is of the form $\langle \mathtt{w} := \mathtt{return} \rangle_i$, and ρ'' is i-matched, ρ'' does not contain more than k contexts of P_i (*scope-bounded matching relations*, SCO for short) [24];
- a (k, d)-*budget execution* ρ is such that for each component program P_i and for each ρ'' such that $\rho = \rho'.\rho''.\rho'''$ where the call stack of P_i contains more than d calls, there are at most k contexts of P_i in ρ'' (*budget-bounded context-switching*, BUD for short) [2][1];
- a *k-phase execution* is the concatenation of k phases where a *phase* of component program P_i is a sequence from Π where all the `return` symbols are from alphabet Π_i, i.e., are of the form $\langle \mathtt{w} := \mathtt{return} \rangle_i$ (*bounded number of phases*, PHA for short) [20];
- an *ordered execution* ρ is such that for each $j \in [n]$ and for each return σ_r from Π_j, all the calls from Π_i, with $i < j$, that occur in ρ before σ_r are matched, i.e., for each decomposition $\rho = \rho'.\sigma_r.\rho''$, ρ' is i-matched for all $i < j$ (*ordered matching relations*, ORD for short) [11];
- a *k-path-tree execution* ρ is such that it can be encoded into a *stack tree*[2] whose nodes can be discovered in the order given by ρ by a walk that starts from the root and visits each node at most k times (*bounded path-trees*, PAT for short) [23].

Bounded reachability. The bounded reachability problem asks to solve reachability by restricting the search within a subset of the coherent program executions that satisfy a given bounding condition. Formally:

▶ **Definition 3** (Bounded Reachability). *Given a concurrent uninterpreted program $\mathcal{P}$ and a bounding condition $\mathcal{B}$ over the executions of $\mathcal{P}$, the $\mathcal{B}$-bounded reachability problem asks whether there exists a feasible and coherent complete execution of $\mathcal{P}$ that satisfies $\mathcal{B}$.*

In the following, we will refer to a $\mathcal{B}$-bounded reachability problem as $\mathcal{B}$-REACH.

Decision algorithm. We reduce the bounded reachability problem to a reachability problem in multistack visibly pushdown automata (MVPA). In particular, we construct an MVPA $\mathcal{A}_{\mathtt{Vars}}$ that captures exactly all the coherent and feasible executions over an alphabet Π, and an MVPA $\mathcal{A}_{\mathcal{P}}$ that captures all the executions of $\mathcal{P}$. We then take the intersection of the two MVPA's and check if it accepts an execution that fulfills the bounding condition. In Section 4, we construct $\mathcal{A}_{\mathtt{Vars}}$ and prove its correctness, and in Section 5 we give $\mathcal{A}_{\mathcal{P}}$ and discuss the correctness and complexity of the decision algorithm for the considered bounding conditions.

[1] The original definition admits a different value of k and d for each component program however the computational complexity of the reachability problem is the same.

[2] A stack tree is a binary tree obtained by labeling the root with the first symbol of ρ, and then the successor in ρ labels the left child unless it is a matched return, and in this case it labels the right child of the matching call.

4 MVPA capturing coherent and feasible executions

In this section, we construct an Mvpa $\mathcal{A}_{\texttt{Vars}}$ that accepts all the coherent and feasible executions of a concurrent uninterpreted program over the variables $\texttt{Vars}$ and functions $\mathcal{F}$.

The crux of the construction is to represent and maintain the equality/disequality/functional (EDF) relationship between variables along a concurrent execution. Concurrency poses new challenges on how to maintain this EDF information. Besides the terms that are currently stored in the program variables, we need to account for those in the local variables of unreturned calls (still in a call stack) for each of the component programs. In concurrent programs, a term can flow from a local variable into a shared variable and then to a local variable of another component program thus potentially establishing direct equality/inequality/superterm relations among the terms stored in two local variables of two different component programs. Moreover, as the execution proceeds, these terms can go deep down into their respective call stacks while no other currently used variable stores them (nor terms that are equivalent to them), and still on returning, these relations need to be restored.

We organize this complex (and unbounded) piece of information into *stack* and *shared states*. Stack states are kept into the stacks of the corresponding component programs while shared states are maintained in the control state of the Mvpa, and both of them store the relations among the content of all variables. To link the shared state with all the stack states at the top of the stacks and a stack state to the next stack state below into the stack, we use *shadow variables*, i.e., additional variables that are used in our relations as placeholders for actual program variables. For each component program we add a shadow variable for each program variable. The stack state of a component program is then augmented with its shadow variables, and the shared state is augmented with the overall set of shadow variables (we need to link this state to all the stacks). When a method call of component P_i is issued, we push into stack i the stack state of P_i that can be derived from the current shared state, and then update the shared state by setting the shadow variables of P_i equal to the corresponding program variables. Shadow variables stay unchanged in all the other cases, and thus we maintain the invariant that a shadow variable of P_i has the value of the corresponding program variable at the time the current method of P_i was called (the initial value in the case of the main method). This way, in each stack, a stack state l is linked to the state l' below it by having each shadow variable of l to evaluate equal to the corresponding program variable in l', thus forming a chain across the stack values. The same holds for the shared state and the stack states at the top of all the stacks.

We recall that a *multistack visibly pushdown automaton* (Mvpa) consists of a finite control along with one or more pushdown stores (stacks) that are driven by the input. We refer the reader to [3, 24] for the details.

In the rest of the section, we first formalize the introduced notions, then we give some details on the construction of the Mvpa and argue its correctness.

Shadow variables. Fix $i \in [n]$. For each component program P_i, we consider a shadow variable for each shared variable and for each variable of all the component programs P_j with $j \in [n]$. We denote the first set as S'_i and the second set as $V'_{i,j}$. Further, we denote $\texttt{Vars}'_i = S'_i \cup \bigcup_{j\in[n]} V'_{i,j}$ (the overall set of shadow variables for P_i), $\texttt{Vars}' = \bigcup_{i\in[n]} \texttt{Vars}'_i$ (the set of all the shadow variables), $\texttt{V} = \texttt{Vars} \cup \texttt{Vars}'$ (the overall set of variables), and $\texttt{V}_i = \texttt{Vars} \cup \texttt{Vars}'_i$ (the set of the program variables along with the shadow variables of P_i).

We extend the notation $\texttt{Comp}$ to capture the described semantics of the shadow variables as follows. For $i \in [n]$, denoting with $x' \in \texttt{Vars}'_i$ the shadow variable corresponding to $x \in \texttt{Vars}$, we set: $\texttt{Comp}(\varepsilon, x') = \texttt{Comp}(\varepsilon, x)$, $\texttt{Comp}(\rho.\langle\texttt{call } m\rangle_i, x') = \texttt{Comp}(\rho, x)$ (i.e., x' stores the value of x on calling a method), $\texttt{Comp}(\rho.\langle\texttt{call}\ m\rangle_i.\rho'.\langle\texttt{w} := \texttt{return}\rangle_i, x') = \texttt{Comp}(\rho, x')$ where ρ' is i-matched (i.e., after the call the previous value of x' is restored) and $\texttt{Comp}(\rho.\sigma, x') = \texttt{Comp}(\rho, x')$ for all $\sigma \notin \{\langle\texttt{call}\ m\rangle_i, \langle\texttt{w} := \texttt{return}\rangle_i \mid m \text{ is a method}\}$.

States and invariants. For an equivalence relation $\sim$ over a set V, we denote with $V/\sim$ the quotient set, i.e., $\{[v]_\sim \mid v \in V\}$.

Given a set of variables V and a set of functions $\mathcal{F}$, let (E, D, P, B) be a tuple such that:

- $E \subseteq V \times V$ is an equivalence relation over V;
- $D \subseteq V/E \times V/E$ is a symmetric relation;
- P is a partial interpretation of the functions from $\mathcal{F}$ over the equivalence classes of E (for an r-ary function f, $P(f)$ is a partial map from $(V/E)^r$ to V/E);
- B is such that for an r-ary function f, $B(f)$ is map from $(V/E)^r$ to $\{\bot, \top\}$.

A *shared state* (resp. *stack state* of P_i for $i \in [n]$) is a tuple of the form (E, D, P, B) as above where $V = \texttt{V}$ (resp. $V = \texttt{V}_i$).

In our construction, along any execution, we aim to maintain a shared state (E, D, P, B) such that E tracks the occurred equivalences, D tracks the occurred inequalities, P captures the superterm relation among the currently stored terms, and B signals that some superterms of the currently stored terms have been already computed. Formally, we wish to maintain the following:

Invariants. For an execution ρ, variables $x, y, x_1, \ldots, x_r \in \texttt{Vars}$ and function $f \in \mathcal{F}$,

I1. $(x, y) \in E$ if and only if $\texttt{Comp}(\rho, x) \cong_{\alpha(\rho)} \texttt{Comp}(\rho, y)$;

I2. $([x]_E, [y]_E) \in D$ if and only if there are $t_0, t_1 \in \texttt{Terms}(\rho)$ s.t. $(t_0, t_1) \in \beta(\rho)$, and for $i \in \{0, 1\}$, $t_i \cong_{\alpha(\rho)} \texttt{Comp}(\rho, x)$ and $t_{1-i} \cong_{\alpha(\rho)} \texttt{Comp}(\rho, y)$;

I3. $P(f)([x_1]_E, \ldots, [x_r]_E) = [x]_E$ if and only if $t \cong_{\alpha(\rho)} f(t_1, \ldots, t_r)$ with $t = \texttt{Comp}(\rho, x)$ and $t_i = \texttt{Comp}(\rho, x_i)$ for $i \in [r]$;

I4. $B(f)([x_1]_E, \ldots, [x_r]_E) = \top$ if and only if there is a prefix ρ' of ρ s.t. $t \cong_{\alpha(\rho)} f(t_1, \ldots, t_r)$ with $t = \texttt{Comp}(\rho', z)$ for some $z \in \texttt{Vars}$ and $t_i = \texttt{Comp}(\rho, x_i)$ for $i \in [r]$.

4.1 The Mvpa $\mathcal{A}_{\texttt{Vars}}$

$\mathcal{A}_{\texttt{Vars}}$ uses n stacks, one for each component program P_i. The symbols of stack i are the stack states. The control states are $\bar{q}_{fs}$, $\bar{q}_{mem}$, $\bar{q}_{ea}$ and the shared states. Intuitively, $\bar{q}_{fs}$, $\bar{q}_{mem}$, and $\bar{q}_{ea}$ are entered when respectively the feasibility, memoizing and early assume property is violated by the input execution. The set of initial states is a singleton containing only the shared state $(E_0, \emptyset, P_0, B_0)$ where $E_0 = \{(x, x) \mid x \in \texttt{V}\}$, and for each $f \in \mathcal{F}$, $P_0(f)$ is undefined and $B_0(f)$ is the constant map assigning $\bot$. Let $\bar{Q}_{fs}$ be the set containing $\bar{q}_{fs}$ and all the shared states (E, D, P, B) such that D is not irreflexive. All the control states are accepting except for $\bar{q}_{mem}$, $\bar{q}_{ea}$ and all the states from $\bar{Q}_{fs}$. The transition relation is defined below.

As we go along with the description of the transitions, we also convey a proof by induction of the fulfillment of the invariants I1–I4 at the control states of the form (E, D, P, B) assuming that the concurrent execution that leads to such states is coherent. Indeed, in our proof we show a stronger property. In fact, we show that invariants I1–I4 hold with respect to $\texttt{V}$ (not only $\texttt{Vars}$) and additionally:

I5. for $i \in [n]$ and any execution of the form $\rho = \rho'.\langle\texttt{call}\ m\rangle_i.\rho''$ where ρ'' is i-matched, the stack state at the top of the stack for program P_i after reading ρ fulfills I1–I4 on ρ'.

The induction is on the length of the input execution. The base case, i.e., when the execution is empty, is a direct consequence of I1–I5 holding at the initial state of $\mathcal{A}_{\texttt{Vars}}$.

Transitions from non-accepting states. The only transitions going out of the states $\bar{q}_{fs}$, $\bar{q}_{mem}$, and $\bar{q}_{ea}$ are transitions to themselves (sink rejecting states). From all states of the form (E, D, P, B) such that D is not irreflexive, it is only possible to reach $\bar{q}_{fs}$.

Internal transitions from accepting states. We describe the internal transitions from a control state q of the form (E, D, P, B) such that D is irreflexive. We start by analyzing two cases that can take to states that are not of the form (E', D', P', B'). On the input symbol $\langle x := f(x_1, \ldots, x_r)\rangle_i$, $\mathcal{A}_{\texttt{Vars}}$ enters $\bar{q}_{mem}$ if $B(f)([x_1]_E, \ldots, [x_r]_E) = \top$ and $P(f)([x_1]_E, \ldots, [x_r]_E)$ is undefined (i.e., when we are trying to recompute a term that is not stored in any variables at this point of the execution and thus the memoizing property breaks). On the input symbol $\langle \texttt{assume}(x = y)\rangle_i$, $\mathcal{A}_{\texttt{Vars}}$ enters $\bar{q}_{ea}$ if there is a superterm t of either the term stored in x or the one stored in y that is stored in z and there is a function $f \in \mathcal{F}$ s.t. $B(f)([x_1]_E, \ldots, [x_r]_E) = \top$ but $P(f)([x_1]_E, \ldots, [x_r]_E)$ is undefined where z is one of $x_1, \ldots, x_r$ (i.e., we get evidence that no term equivalent to a previously computed superterm of those stored in either x or y is currently stored, and thus the early assume property breaks).

In the remaining cases, the internal transitions take to a state of the form (E', D', P', B') such that on input σ and with $i \in [n]$: if $\sigma = \langle \texttt{assume}(x = y)\rangle_i$, we merge the equivalence classes of x and y and propagate equality on the stored superterms (by P), then update D, P and B according to the equivalence classes of E'; if $\sigma = \langle \texttt{assume}(x \neq y)\rangle_i$, we just add $([x]_E, [y]_E)$ and $([y]_E, [x]_E)$ to the set of inequalities D; if $\sigma = \langle x := f(x_1, \ldots, x_r)\rangle_i$, we essentially move x to the equivalence class $P(f)([x_1]_E, \ldots, [x_r]_E)$ if defined and start a new one otherwise, remove the pairs of D involving x if x was the only variable of its class, and update P and B according to E'; the case $\sigma = \langle x := y\rangle_i$ is simpler than the previous one, we just need to remove the pairs of D involving x if x was the only variable of its class, merge the equivalence classes of x and y, and modify P and B accordingly. It is simple to see that the invariants I1–I5 are preserved.

Push and pop transitions. The only push and pop transitions are from control states of the form (E, D, P, B). As for the internal transitions, (E, D, P, B) is updated with the purpose of preserving the wished invariants. Additionally, on a call symbol of the form $\langle \texttt{call}\ m\rangle_i$, the shadow variables of component program P_i are set to the terms currently stored in the corresponding program variables (by enforcing equality with these variables) and the current stack state (i.e., the restriction of the current control state to the variables $\texttt{V}_i$) is pushed onto stack i. On a return symbol of the form $\langle \texttt{w} := \texttt{return}\rangle_i$, the stack state at the top of stack i is popped and merged with the current control state such that the resulting state relates: the terms at the beginning of the resumed method call (referred by the variables from $\texttt{Vars}'_i$ in the popped state) to the rest of the terms of the current state (referred by the variables other than $\texttt{Vars}'_i$ in the current control state). Below, we give the details only for the return transitions.

Let $\sigma = \langle \texttt{w} := \texttt{return}\rangle_i$ be the input symbol, $q = (E, D, P, B)$ be the current control state, m be the returned method and $q_\ell = (E_\ell, D_\ell, P_\ell, B_\ell)$ be the top symbol of stack i. From q, by reading σ and popping q_ℓ, $\mathcal{A}_{\texttt{Vars}}$ moves to a control state $q' = (E', D', P', B')$ that is obtained as follows. We start by renaming in q_ℓ each variable x to $\underline{x}$. Denote $\underline{\texttt{Vars}} = \{\underline{x} \mid x \in \texttt{Vars}\}$ and $\underline{\texttt{Vars}'_i} = \{\underline{x}' \mid x' \in \texttt{Vars}'_i\}$ (recall that q_ℓ is over the set of variables $\texttt{V}_i$). Then, we define q'' as the component-wise union of q and q_ℓ (i.e., we retain the equivalences, inequalities and map definitions from these states). Note that q'' is over the variables from $\texttt{V} \cup \underline{\texttt{Vars}} \cup \underline{\texttt{Vars}'_i}$.

Now, we update q'' by assuming the equations $x' = \underline{x}$ for $x \in \mathtt{Vars}$ where $x' \in \mathtt{Vars}'_i$ and $\underline{x} \in \underline{\mathtt{Vars}}$ are the variables corresponding to x in the respective sets. In the resulting state we then drop all the variables from $\mathtt{Vars}'_i \cup \underline{\mathtt{Vars}}$ and rename back each variable $\underline{x}' \in \underline{\mathtt{Vars}'_i}$ to the corresponding variable $x' \in \mathtt{Vars}'_i$. The resulting state q_{mrg} is thus over the variables from $\mathtt{V}$. Finally, we get q' by updating this state according to the assignments $\mathtt{w} := \mathtt{o}_m$.

To argue the induction step in this case, let $\rho = \rho'.\langle \mathtt{call}\ m\rangle_i.\rho''$ where ρ'' is i-matched. From the induction hypothesis, q_ℓ fulfills I1–I4 restricted to $\mathtt{V}_i$ on ρ' (I5) and q fulfills I1–I4 on ρ. Thus, by effect of the equations $x' = \underline{x}$ that we assumed to get q_{mrg} from q'', we relate the valuation of each variable at the end of ρ' to: its valuation at the end of ρ for the variables listed in $\mathtt{w}$ (i.e., before assigning the terms returned by method m), and its valuation at the end of $\rho.\langle \mathtt{w} := \mathtt{return}\rangle_i$ for the remaining ones. Therefore, the state q_{mrg} fulfills I1–I4 on $\rho.\langle \mathtt{w}_m := \mathtt{return}\rangle_i$ except for assuming the valuation $\mathtt{Comp}(\rho, w)$ for w in $\mathtt{w}$. Finally, since q' is obtained from q_{mrg} through the assignments $\mathtt{w} := \mathtt{o}_m$, we get that q' fulfills I1–I4 on $\rho.\langle \mathtt{w}_m := \mathtt{return}\rangle_i$. Further, after the transition is taken, the stack state at the top of the stack must clearly fulfill I5. In fact, since on reading a call of P_i we push onto the corresponding stack the restriction of the control state to $\mathtt{V}_i$, by the inductive hypothesis we get that I1–I4 clearly holds up to that point of the computation.

Correctness. From the above arguments, the following lemma holds:

▶ **Lemma 4.** *Let ρ be a coherent concurrent execution over variables* $\mathtt{Vars}$ *and functions $\mathcal{F}$. If $\mathcal{A}_{\mathtt{Vars}}$ reaches a control state of the form (E, D, P, B) after reading ρ, then (E, D, P, B) satisfies the invariants I1–I4.*

By building on the results from [27] and the above lemma, we get:

▶ **Lemma 5.** *Let ρ be a coherent concurrent execution over variables* $\mathtt{Vars}$ *and functions $f \in \mathcal{F}$. For $\sigma \in \Pi$, the following holds:*

1. *ρ is infeasible iff $\mathcal{A}_{\mathtt{Vars}}$ enters a state in $\bar{Q}_{fs}$ on input ρ;*
2. *$\rho.\sigma$ is not memoizing iff $\mathcal{A}_{\mathtt{Vars}}$ enters $\bar{q}_{mem}$ on input $\rho.\sigma$;*
3. *$\rho.\sigma$ does not satisfy the early-assumes property iff $\mathcal{A}_{\mathtt{Vars}}$ enters $\bar{q}_{ea}$ on input $\rho.\sigma$.*

Since the only non-accepting states of $\mathcal{A}_{\mathtt{Vars}}$ are $\bar{q}_{mem}$, $\bar{q}_{ea}$ and all the states from $\bar{Q}_{fs}$, by inductively applying the above lemma we get:

▶ **Theorem 6.** *A concurrent computation ρ is accepted by $\mathcal{A}_{\mathtt{Vars}}$ if and only if ρ is coherent and feasible.*

By assuming that the signature has constant size, the number of different tuples of the form (E, D, P, B) over the set of variables $\mathtt{V}$ is $O(2^{|\mathtt{V}|^{O(1)}})$ where $|\mathtt{V}| = O(n\,|\mathtt{Vars}|)$, and thus, also the size of $\mathcal{A}_{\mathtt{Vars}}$ is $\mathtt{V}$ is $O(2^{|\mathtt{V}|^{O(1)}})$.

5 Checking bounded reachability and coherence

Fix a concurrent uninterpreted program $\mathcal{P}$ with component programs $P_1, \ldots, P_n$.

Reduction to Mvpa reachability. By standard constructions, it is possible to construct an Mvpa $\mathcal{A}_{\mathcal{P}}$ of size exponential in the number of components n that accepts all and only the complete executions of $\mathcal{P}$.

Since the stack operations are visible in the input alphabet, the intersection of two Mvpas is still an Mvpa that can be obtained by the cross product of the starting Mvpas [3, 24]. Denoting $\mathcal{A}_{\mathcal{P},\mathtt{Vars}}$ the Mvpa capturing the intersection of $\mathcal{A}_{\mathcal{P}}$ and $\mathcal{A}_{\mathtt{Vars}}$, the size of $\mathcal{A}_{\mathcal{P},\mathtt{Vars}}$ is $2^{|\mathtt{V}|^{O(1)}}$ (note that $|\mathtt{V}| \geq n$). Thus, by Theorem 6, we have:

▶ **Theorem 7.** *$\mathcal{A}_{\mathcal{P},\mathtt{Vars}}$ has size $2^{|\mathtt{V}|^{O(1)}}$ and accepts all and only the coherent and feasible executions of $\mathcal{P}$.*

Decidable bounded reachability problems. By restricting the executions with the bounding conditions given in Section 3, we obtain versions of MVPA's that have a decidable reachability problem. This along with Theorem 7 gives the decidability of the bounded reachability problem for concurrent uninterpreted programs under all the considered bounding conditions. Concerning to the computational complexity, we have the following upper-bounds (we denote with A an n-stack MVPA):

- from [30], the MVPA reachability problem within k context-switches can be solved in time $O(k^3|A|^5(n|Q|)^k)$ where Q denotes the set of control states of A, thus by Theorem 7 we get that CON-REACH can be decided in time exponential in the size of $\mathtt{V}$ and k;
- from [24] we have that the MVPA reachability problem restricted to k-scoped executions can be decided in $O(2^n|Q|^{2kn+1})$ time, and thus by Theorem 7 we get that SCO-REACH can be decided in time exponential in the size of $\mathtt{V}$ and k;
- from [2], we have that the MVPA reachability problem restricted to (k,d)-budget executions can be decided in time exponential in $(|A|+d+k)$; by encoding the stacks up to depth d into the control state, we can give a decision algorithm in the style of that given for the scope-bounded restriction that takes $O(2^n(|Q|+|\Gamma|^{nd})^{2kn+1})$ time, which gives for BUD-REACH an upper-bound that is exponential in size of $\mathtt{V}$, d and k;
- from [23], we have that the MVPA reachability problem restricted to k-path-tree executions can be decided in time $2^{O(k(n+\log|A|))}$, thus by Theorem 7 we get that PAT-REACH can be decided in time exponential in the size of $\mathtt{V}$ and k;
- from [20], we have that the MVPA reachability problem restricted to k-phase executions can be decided in time $2^{|A|2^{O(k)}}$, thus by Theorem 7 we get that PHA-REACH can be decided in time double exponential in the size of $\mathtt{V}$ and k;
- from [4], we have that the MVPA reachability problem restricted to ordered executions can be decided in time $|A|^{2^{O(n)}}$, thus by Theorem 7 we get that ORD-REACH can be decided in time exponential in the size of $\mathtt{V}$ and double exponential in n.

Since the reachability problem for sequential uninterpreted programs is EXPTIME-hard [27] and each instance of this problem is also an instance of CON-REACH, SCO-REACH, BUD-REACH and PAT-REACH, we have that all these problems are EXPTIME-complete. Moreover, since the reachability of MVPA restricted to ordered executions and bounded phase executions can be reduced to the respective problems for Boolean programs, and both these problems are 2EXPTIME-hard [4, 20], we have that PHA-REACH and ORD-REACH are 2EXPTIME-complete. Thus, we get the following theorem:

▶ **Theorem 8.** *The problems CON-REACH, SCO-REACH, BUD-REACH and PAT-REACH are EXPTIME-complete, and the problems PHA-REACH and ORD-REACH are 2EXPTIME-complete.*

Deciding coherence. Define $\mathcal{A}_{notco}$ as the MVPA obtained from $\mathcal{A}_{\mathtt{Vars}}$ by removing the state $\bar{q}_{fs}$ and the transitions involving it, and making $\bar{q}_{mem}$ and $\bar{q}_{ea}$ its only accepting states. From Lemma 5, we get that $\mathcal{A}_{notco}$ accepts a concurrent execution ρ if and only if ρ is incoherent. Thus, to determine whether a program $\mathcal{P}$ is coherent we can just check $\mathcal{A}_{notco} \cap \mathcal{A}_{\mathcal{P}} = \emptyset$. Therefore, by the results on the considered bounding conditions introduced above, we get:

▶ **Theorem 9.** *Deciding coherence is EXPTIME-complete under CON, SCO, BUD and PAT restrictions and 2EXPTIME-complete under PHA and ORD restrictions.*

6 Conclusions

In this paper, we have shown the decidability of the reachability problem for concurrent uninterpreted programs under a number of restrictions that have been considered in the literature for the analysis of finite-domain concurrent programs. Our results do not extend directly to parametric uninterpreted programs, i.e., concurrent uninterpreted programs with executions formed of unboundedly many component programs (see [5, 21, 16, 22]). In fact, we crucially use in our reduction to Mvpa to distinguish among the local variables of each component program. Also, known results on sequentializations (see [19, 25, 31]), i.e., code-to-code translations into non-deterministic sequential programs which (under certain assumptions) behave equivalently, do not seem to to work for uninterpreted programs as coherence suddenly breaks when rearranging the order of the statements. Both these directions deserve future investigation.

References

1 Parosh Aziz Abdulla, C. Aiswarya, and Mohamed Faouzi Atig. Data Multi-Pushdown Automata. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 38:1–38:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.38.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Othmane Rezine, and Jari Stenman. Budget-bounded model-checking pushdown systems. *Formal Methods in System Design*, 45(2):273–301, 2014. doi:10.1007/s10703-014-0207-y.

3 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009. doi:10.1145/1516512.1516518.

4 Mohamed Faouzi Atig, Benedikt Bollig, and Peter Habermehl. Emptiness of Ordered Multi-Pushdown Automata is 2ETIME-Complete. *Int. J. Found. Comput. Sci.*, 28(8):945–976, 2017. doi:10.1142/S0129054117500332.

5 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-Bounded Analysis For Concurrent Programs With Dynamic Creation of Threads. *Logical Methods in Computer Science*, 7(4), 2011. doi:10.2168/LMCS-7(4:4)2011.

6 Mikolaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.

7 Mikolaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13:1–13:48, 2009. doi:10.1145/1516512.1516515.

8 Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. Model Checking Languages of Data Words. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7213 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 2012. doi:10.1007/978-3-642-28729-9_26.

9 Ahmed Bouajjani, Michael Emmi, and Gennaro Parlato. On Sequentializing Concurrent Programs. In Eran Yahav, editor, *Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings*, volume 6887 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2011. doi:10.1007/978-3-642-23702-7_13.

10 Aaron R. Bradley and Zohar Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification.* Springer-Verlag, Berlin, Heidelberg, 2007.

11 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-Push-Down Languages and Grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.

12 Denis Bueno and Karem A. Sakallah. euforia: Complete Software Model Checking with Uninterpreted Functions. In Constantin Enea and Ruzica Piskac, editors, *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings*, volume 11388 of *Lecture Notes in Computer Science*, pages 363–385. Springer, 2019. doi:10.1007/978-3-030-11245-5_17.

13 Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012. doi:10.1007/978-3-642-32940-1_38.

14 Claire David, Leonid Libkin, and Tony Tan. On the Satisfiability of Two-Variable Logic over Data Words. In Christian G. Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings*, volume 6397 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2010. doi:10.1007/978-3-642-16242-8_18.

15 Martin Davis. Kurt Gödel. Über Die Vollständigkeit des Logikkalküls . Collected Works, Volume I, Publications 1929–1936, by Kurt Gödel, Edited by Solomon Feferman, John W. Dawson Jr., Stephen C. Kleene, Gregory H. Moore, Robert M. Solovay, and Jean van Heijenoort, Clarendon Press, Oxford University Press, New York and Oxford, 1986, Even Pp. 60– 100. - Kurt Gödel. On the Completeness of the Calculus of Logic . English Translation by Stefan Bauer-Mengelberg and Jean van Heijenoort of the Preceding. *Journal of Symbolic Logic*, 55(1):341–342, 1990. doi:10.2307/2274974.

16 Antoine Durand-Gasselin, Javier Esparza, Pierre Ganty, and Rupak Majumdar. Model checking parameterized asynchronous shared-memory systems. *Formal Methods in System Design*, 50(2-3):140–167, 2017. doi:10.1007/s10703-016-0258-3.

17 Michael Emmi, Shaz Qadeer, and Zvonimir Rakamaric. Delay-bounded scheduling. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 411–422. ACM, 2011. doi:10.1145/1926385.1926432.

18 Sumit Gulwani and Ashish Tiwari. Assertion Checking Unified. In Byron Cook and Andreas Podelski, editors, *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, volume 4349 of *Lecture Notes in Computer Science*, pages 363–377. Springer, 2007. doi:10.1007/978-3-540-69738-1_26.

19 Omar Inverso, Ermenegildo Tomasco, Bernd Fischer, Salvatore La Torre, and Gennaro Parlato. Bounded Model Checking of Multi-threaded C Programs via Lazy Sequentialization. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 585–602. Springer, 2014. doi:10.1007/978-3-319-08867-9_39.

20 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. A Robust Class of Context-Sensitive Languages. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 161–170. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.9.

21 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Model-Checking Parameterized Concurrent Programs Using Linear Interfaces. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 629–644. Springer, 2010. doi:10.1007/978-3-642-14295-6_54.

22 Salvatore La Torre, Anca Muscholl, and Igor Walukiewicz. Safety of Parametrized Asynchronous Shared-Memory Systems is Almost Always Decidable. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 72–84. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.72.

23 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. A Unifying Approach for Multistack Pushdown Automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 377–389. Springer, 2014. doi:10.1007/978-3-662-44522-8_32.

24 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Scope-Bounded Pushdown Languages. *Int. J. Found. Comput. Sci.*, 27(2):215–234, 2016. doi:10.1142/S0129054116400074.

25 Akash Lal and Thomas W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. *Formal Methods in System Design*, 35(1):73–97, 2009. doi:10.1007/s10703-009-0078-9.

26 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 283–294. ACM, 2011. doi:10.1145/1926385.1926419.

27 Umang Mathur, P. Madhusudan, and Mahesh Viswanathan. Decidable verification of uninterpreted programs. *PACMPL*, 3(POPL):46:1–46:29, 2019. URL: https://dl.acm.org/citation.cfm?id=3290359, doi:10.1145/3290359.

28 Umang Mathur, Adithya Murali, Paul Krogmeier, P. Madhusudan, and Mahesh Viswanathan. Deciding Memory Safety for Forest Datastructures. *CoRR*, abs/1907.00298, 2019. arXiv:1907.00298.

29 Andrzej S. Murawski, Steven J. Ramsay, and Nikos Tzevelekos. Reachability in pushdown register automata. *J. Comput. Syst. Sci.*, 87:58–83, 2017. doi:10.1016/j.jcss.2017.02.008.

30 Shaz Qadeer and Jakob Rehof. Context-Bounded Model Checking of Concurrent Software. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005. doi:10.1007/978-3-540-31980-1_7.

31 Shaz Qadeer and Dinghao Wu. KISS: keep it simple and sequential. In William Pugh and Craig Chambers, editors, *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation 2004, Washington, DC, USA, June 9-11, 2004*, pages 14–24. ACM, 2004. doi:10.1145/996841.996845.

32 Nikos Tzevelekos. Fresh-register automata. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 295–306. ACM, 2011. doi:10.1145/1926385.1926420.

Distance Between Mutually Reachable Petri Net Configurations

Jérôme Leroux
LaBRI, CNRS, Univ. Bordeaux, France
jerome.leroux@labri.fr

Abstract

Petri nets are a classical model of concurrency widely used and studied in formal verification with many applications in modeling and analyzing hardware and software, data bases, and reactive systems. The reachability problem is central since many other problems reduce to reachability questions. In 2011, we proved that a variant of the reachability problem, called the reversible reachability problem is exponential-space complete. Recently, this problem found several unexpected applications in particular in the theory of population protocols. In this paper we revisit the reversible reachability problem in order to prove that the minimal distance in the reachability graph of two mutually reachable configurations is linear with respect to the Euclidean distance between those two configurations.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Petri nets, Vector addition systems, Formal verification, Reachability problem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.47

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-02156549`.

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

1 Introduction

Petri nets are a classical model of concurrency widely used and studied in formal verification with many applications in modeling and analyzing hardware and software, data bases, and reactive systems. The reachability problem is central since many other problems reduce to reachability questions. Unfortunately, the reachability problem is difficult for several reasons. In fact, from a complexity point of view, we recently proved that the problem is non-elementary [6] by observing that the worst case complexity in space is at least a tower of exponential with height growing linearly in the dimension of the Petri nets. Moreover, even in practice, the reachability problem is difficult. Nowadays, no tool exists for deciding that problem since the known algorithms are difficult to be implemented and require many enumerations in exponentially large state spaces (see [13] for the state-of-the-art algorithm deciding the reachability problem).

Fortunately, easier natural variants of the reachability problems can be applied in various contexts. For instance, the coverability problem which consists in deciding if a configuration can be covered by a reachable one can be applied in the analysis of concurrent programs [1] (in that context, covered means component-wise smaller than or equal). The coverability problem is known to be exponential-space complete [16, 5], and efficient tools exist [4, 8]. Another variant is the reversible reachability problem. This problem consists in deciding if two configurations are mutually reachable one from the other. This problem was proved to be exponential-space complete in [11] and finds unexpected applications in population protocols [7], trace logics [12], universality problems related to structural liveness problems [10], and in solving the home state problem [2].

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 47; pp. 47:1–47:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Contribution. The exponential-space complexity upper-bound of the reversible reachability problem proved in [11] is obtained by observing that if two configurations are mutually reachable, then the two configurations belong to a cycle of the (infinite) reachability graph with a length at most doubly-exponential with respect to the size in binary of the two configurations. In this paper, we focus on the minimal length of such a cycle (called the distance in the sequel) with respect to the Euclidean distance between those two configurations. We prove that the distance is linearly bounded by the Euclidean distance up-to a doubly-exponential constant that only depends on the Petri net. As a direct consequence, this result generalizes [11] and it shows that the distance between two nearby (for the Euclidean distance) mutually reachable configurations is small.

Outline. In Section 2 we introduce our main problem about the distance between mutually reachable Petri net configurations and we motivate the problem. In Section 3 we extend Petri nets with control states. A petri net with states is basically given as a finite state automaton with transitions labeled by Petri net actions. We also introduce the subclass of structurally reversible Petri nets. Intuitively a Petri net with states is structurally reversible if the effect of every transition can be reverted as soon as we execute that transition from a large enough configuration. We provide in that section a sufficient condition to decide the reachability problem for structurally reversible Petri nets between large configurations. In Section 4 and Section 5, we recall some techniques called *Rackoff's extraction* to extract components that are "very large" compared to others from executions. Those techniques are applied in Section 6 in order to extract from a strongly connected component of the reachability graph of a Petri net, a structurally reversible Petri net with states. Intuitively, this Petri net with states is obtained by projecting away components that can be large in the considered strongly connected component. From that Petri net with states, and thanks to the sufficient condition for reachability introduced in Section 3, we proved in Section 7 our main result about the distance between mutually reachable configurations.

In the paper, d is a positive natural number denoting the number of components of vectors. Given a vector $\mathbf{x}$ in the set of reals $\mathbb{R}^d$, we denote by $\mathbf{x}(1), \ldots, \mathbf{x}(d)$ its components in such a way $\mathbf{x} = (\mathbf{x}(1), \ldots, \mathbf{x}(d))$. Moreover, we introduce the norms $\|\mathbf{x}\| \overset{\text{def}}{=} \sum_{i=1}^{d} |\mathbf{x}(i)|$ and $\|\mathbf{x}\|_\infty \overset{\text{def}}{=} \max_{1 \leq i \leq d} |\mathbf{x}(i)|$. The set of integers and the set of non-negative natural numbers are denoted as $\mathbb{Z}$ and $\mathbb{N}$ respectively.

2 Petri Nets

A *Petri net* A (*PN* for short) is a finite set of pairs $(\mathbf{a}_-, \mathbf{a}_+)$ in $\mathbb{N}^d \times \mathbb{N}^d$ called *actions*. In the literature, vectors $\mathbf{a}_-$ and $\mathbf{a}_+$ are respectively usually called the *pre-condition* and the *post-condition* of a. A *configuration* is a vector in $\mathbb{N}^d$. We associate to an action $a = (\mathbf{a}_-, \mathbf{a}_+)$ the binary relation $\xrightarrow{a}$ over the configurations defined by $\mathbf{x} \xrightarrow{a} \mathbf{y}$ if for some configuration $\mathbf{c}$ we have $\mathbf{x} = \mathbf{a}_- + \mathbf{c}$ and $\mathbf{y} = \mathbf{a}_+ + \mathbf{c}$. Notice that $\mathbf{x} \xrightarrow{a} \mathbf{y}$ if, and only if, $\mathbf{x} \geq \mathbf{a}_-$ and $\mathbf{y} = \mathbf{x} - \mathbf{a}_- + \mathbf{a}_+$. We denote by $\xrightarrow{A}$ the one-step reachability relation of A defined by $\mathbf{x} \xrightarrow{A} \mathbf{y}$ if there exists an action a in A such that $\mathbf{x} \xrightarrow{a} \mathbf{y}$. A PN A defines an infinite graph $(\mathbb{N}^d, \xrightarrow{A})$ called the *reachability graph* of A.

A *σ-execution*, where $\sigma = a_1 \ldots a_k$ is a word of actions, is a non-empty word of configurations $e = \mathbf{c}_0\mathbf{c}_1 \ldots \mathbf{c}_k$ such that the following relations hold:

$$\mathbf{c}_0 \xrightarrow{a_1} \mathbf{c}_1 \cdots \xrightarrow{a_k} \mathbf{c}_k$$

We denote by $\mathrm{src}(e)$ and $\mathrm{tgt}(e)$ the configurations $\mathbf{c}_0$ and $\mathbf{c}_k$ respectively. An A^*-*execution* is a σ-execution for some word σ over A. An A^ω-*execution* e is an infinite word of configurations such that every finite non-empty prefix is an A^*-execution. We associate to a word σ of actions the binary relation $\xrightarrow{\sigma}$ over the configurations defined by $\mathbf{x} \xrightarrow{\sigma} \mathbf{y}$ if there exists a σ-execution from $\mathbf{x}$ to $\mathbf{y}$. The *displacement* of a word $\sigma = a_1 \dots a_k$ is the vector $\Delta(\sigma) \stackrel{\text{def}}{=} \sum_{j=1}^k \Delta(a_j)$ where $\Delta(a) \stackrel{\text{def}}{=} \mathbf{a}_+ - \mathbf{a}_-$ for every action $a = (\mathbf{a}_-, \mathbf{a}_+)$. Notice that $\mathbf{x} \xrightarrow{\sigma} \mathbf{y}$ implies $\Delta(\sigma) = \mathbf{y} - \mathbf{x}$ but the converse is not true in general. We introduce the *reachability relation* $\xrightarrow{A^*}$ defined as the union of the relations $\xrightarrow{\sigma}$ where $\sigma \in A^*$. Notice that this relation is the reflexive and transitive closure of $\xrightarrow{A}$.

The Petri net reachability problem consists in deciding given a PN A and two configurations $\mathbf{x}$ and $\mathbf{y}$ if $\mathbf{x} \xrightarrow{A^*} \mathbf{y}$. In [6], we provided a non-elementary complexity lower-bound for the PN reachability problem. Moreover, we prove that for every natural number h, there exists a PN A_h such that the reachability problem for that PN is at least h-exponential space hard. It means that the minimal length of a word $\sigma \in A_h^*$ satisfying $\mathbf{x} \xrightarrow{\sigma} \mathbf{y}$ is at least $(h+1)$-exponential with respect to $\|\mathbf{x}\| + \|\mathbf{y}\|$. This huge lower bound is no longer valid for mutually reachable configurations.

Two configurations $\mathbf{x}$ and $\mathbf{y}$ are said to be *mutually reachable* for a PN A if $\mathbf{x} \xrightarrow{A^*} \mathbf{y}$ and $\mathbf{y} \xrightarrow{A^*} \mathbf{x}$. The PN reversible reachability problem consists in deciding given a PN A and two configurations $\mathbf{x}$ and $\mathbf{y}$ if they are mutually reachable for A. In [11], we proved that the PN reversible reachability problem is decidable in exponential-space by proving that there exists at most doubly-exponential long words σ and w in A^* such that $\mathbf{x} \xrightarrow{\sigma} \mathbf{y}$ and $\mathbf{y} \xrightarrow{w} \mathbf{x}$. This result can be refined by introducing the notion of distance. The *distance* between two mutually reachable configurations $\mathbf{x}$ and $\mathbf{y}$ for a PN A is formally defined as follows:

$$\mathrm{dist}_A(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} \min_{\sigma, w \in A^*} \{|\sigma w| \mid \mathbf{x} \xrightarrow{\sigma} \mathbf{y} \xrightarrow{w} \mathbf{x}\}$$

A simple lower bound on the distance can be obtained by observing that configurations along an execution are relatively close one from each other as shown in the proof of the following lemma.

▶ **Lemma 1.** *Let us consider a PN $A \subseteq \{0, \dots, m\}^d \times \{0, \dots, m\}^d$ for some positive natural number m. For every mutually reachable configurations $\mathbf{x}$ and $\mathbf{y}$, we have:*

$$\mathrm{dist}_A(\mathbf{x}, \mathbf{y}) \geq \|\mathbf{y} - \mathbf{x}\| \frac{2}{dm}$$

Proof. Let σ be a word in A^* such that $\mathbf{x} \xrightarrow{\sigma} \mathbf{y}$ and let us prove that $\|\mathbf{y} - \mathbf{x}\| \leq m|\sigma|$. Assume that $\sigma = a_1 \dots a_k$. Since $\Delta(a_j) \in \{-m, \dots, m\}^d$, it follows that $\Delta(\sigma) \in \{-mk, \dots, mk\}^d$. In particular $\|\Delta(\sigma)\| \leq dmk$. As $\Delta(\sigma) = \mathbf{y} - \mathbf{x}$ and $k = |\sigma|$, we deduce the relation $\|\mathbf{y} - \mathbf{x}\| \leq md|\sigma|$. Now, let us consider a word w in $\mathbf{A}^*$ such that $\mathbf{y} \xrightarrow{w} \mathbf{x}$ and observe that we have $\|\mathbf{x} - \mathbf{y}\| \leq dm|w|$ by symmetry. It follows that $|\sigma w| \geq \|\mathbf{y} - \mathbf{x}\| \frac{2}{dm}$ and we have proved the lemma. ◀

This paper focus on an upper-bound of the form $\mathrm{dist}_A(\mathbf{x}, \mathbf{y}) \leq f_A(\|\mathbf{y} - \mathbf{x}\|)$ where f_A is a function that only depends on the PN A and not on the two mutually reachable configurations $\mathbf{x}$ and $\mathbf{y}$. Such a bound cannot be derived from [11]. In fact, the best upper bound that can be derived from that paper is the following one:

$$\mathrm{dist}_A(\mathbf{x}, \mathbf{y}) \leq 34 d^2 n^{15 d^{d+2}}$$

where $n = (1 + 2m)(1 + 2\max\{\|\mathbf{x}\|, \|\mathbf{y}\|\})$.

In this paper we prove that such a function f_A exists. Moreover a linear one exists as shown by the following theorem.

▶ **Theorem 2.** *Let us consider a PN $A \subseteq \{0, \dots, m\}^d \times \{0, \dots, m\}^d$ for some positive natural number m. For every mutually reachable configurations $\mathbf{x}$ and $\mathbf{y}$, we have:*

$$\text{dist}_A(\mathbf{x}, \mathbf{y}) \leq \|\mathbf{y} - \mathbf{x}\| c_{d,m}$$

where:

$$c_{d,m} \leq (3dm)^{(d+1)^{2d+4}}$$

▶ Remark 3. The previous theorem provides as a corollary a new proof that the reversible reachability problem is solvable in exponential space. It also provides a bound on the minimal elements defining the *domain of reversibility* (introduced in [11]) of an action a in A defined as $\mathbf{D}_{a,A} \overset{\text{def}}{=} \{\mathbf{x} \in \mathbb{N}^d \mid \exists \mathbf{y}\ \mathbf{x} \xrightarrow{a} \mathbf{y} \xrightarrow{A^*} \mathbf{x}\}$. In fact, this set is *upward closed* and if $\mathbf{x}$ is a minimal element for $\leq$ in $\mathbf{D}_{a,A}$ then the vector $\mathbf{y}$ satisfying $\mathbf{x} \xrightarrow{a} \mathbf{y}$ is such that $\text{dist}_A(\mathbf{x}, \mathbf{y}) \leq dmc_{d,m}$ since $\|\mathbf{y} - \mathbf{x}\| = \|\Delta(a)\| \leq dm$. We deduce that there exists a word σ of actions in A such that $\mathbf{y} \xrightarrow{\sigma} \mathbf{x}$ with a length bounded by $dmc_{d,m}$. If a component of $\mathbf{x}$ is larger than $m|\sigma|$, the vector $\mathbf{x}$ cannot be minimal since the vector $\mathbf{x}'$ obtained from $\mathbf{x}$ by replacing that coordinate by $m|\sigma|$ satisfies $\mathbf{x}' \xrightarrow{a} \mathbf{y}' \xrightarrow{\sigma} \mathbf{x}'$ where $\mathbf{y}' \overset{\text{def}}{=} \mathbf{x}' + \Delta(a)$. Hence $\|\mathbf{x}\| \leq dm^2 c_{d,m}$.

3 Structurally Reversible Petri Nets With States

A *Petri net with states* (*PNS* for short) is a tuple $\langle Q, A, T \rangle$ where Q is a non empty finite set of elements called *states*, A is a Petri net, and T is a set of triples in $Q \times A \times Q$ called *transitions*. A *path* π from a state p to a state q labeled by a word σ of actions is a word of transitions of the form $(q_0, a_1, q_1) \dots (q_{k-1}, a_k, q_k)$ for some states $q_0, \dots, q_k$ satisfying $q_0 = p$ and $q_k = q$, and for some actions $a_1, \dots, a_k$ satisfying $\sigma = a_1 \dots a_k$. The *displacement* of π is the vector $\Delta(\pi) \overset{\text{def}}{=} \Delta(\sigma)$. A path is said to be *elementary* if $q_i = q_j$ implies $i = j$. A path such that $q_0 = q_k$ is called a *cycle* on q_0. A cycle is said to be *simple* if $q_i = q_j$ with $i < j$ implies $i = 0$ and $j = k$. A pair $(q, \mathbf{x})$ in $Q \times \mathbb{N}^d$ is called a *state-configuration* and it is denoted as $q(\mathbf{x})$ in the sequel. We associate to a path π the binary relation $\xrightarrow{\pi}$ over the state-configurations defined by $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ if π is a path from p to q labeled by a word σ of actions such that $\mathbf{x} \xrightarrow{\sigma} \mathbf{y}$.

A PNS is said to be *structurally reversible* if for every transition (p, a, q) there exists a path π from q to p such that $\Delta(a) + \Delta(\pi) = \mathbf{0}$. Structurally reversible PNSes are such that the displacement of any cycle can be canceled by the displacement of another cycle as shown by the following lemma.

▶ **Lemma 4.** *For every state q and for every cycle θ on q, there exists a cycle θ' on q such that $\Delta(\theta') = -\Delta(\theta)$.*

Proof. Assume that $\theta = (q_0, a_1, q_1) \dots (q_{k-1}, a_k, q_k)$ with $q_0 = q = q_k$. Since the PNS is structurally reversible, for every $j \in \{1, \dots, k\}$, there exists a path π_j from q_j to q_{j-1} such that $\Delta(a_j) + \Delta(\pi_j) = \mathbf{0}$. Now, observe that $\theta' \overset{\text{def}}{=} \pi_k \dots \pi_1$ is a cycle on q such that $\Delta(\theta') = -\Delta(\theta)$. ◀

A *partial configuration* is a vector $\mathbf{x} \in \mathbb{N}^I$ where $I \subseteq \{1, \ldots, d\}$. We associate to a configuration $\mathbf{x} \in \mathbb{N}^d$ and a set $I \subseteq \{1, \ldots, d\}$ the partial configuration $\mathbf{x}|_I$ in $\mathbb{N}^I$ defined by $\mathbf{x}|_I(i) = \mathbf{x}(i)$ for every $i \in I$. Given an action $a = (\mathbf{a}_-, \mathbf{a}_+)$ of a Petri net, we extend the binary relation $\xrightarrow{a}$ over the partial configurations by $\mathbf{x} \xrightarrow{a} \mathbf{y}$ if $\mathbf{x}, \mathbf{y}$ are two partial configurations in $\mathbb{N}^I$ such that there exists a partial configuration $\mathbf{c} \in \mathbb{N}^I$ satisfying $\mathbf{x} = \mathbf{a}_-|_I + \mathbf{c}$ and $\mathbf{y} = \mathbf{a}_+|_I + \mathbf{c}$.

A *flow function* is a function $F : Q \to \mathbb{N}^I$ for some subset $I \subseteq \{1, \ldots, d\}$ such that $F(p) \xrightarrow{a} F(q)$ for every transition (p, a, q) in T. In this section we prove the following result.

▶ **Lemma 5.** *Let us consider a structurally reversible PNS with at most r states and with actions in $\{0, \ldots, m\}^d \times \{0, \ldots, m\}^d$ for some positive natural number m, let $p(\mathbf{x})$ and $q(\mathbf{y})$ be two state-configurations such that the following conditions hold for some flow function $F : Q \to \mathbb{N}^I$:*

- *$\mathbf{x}|_I = F(p)$ and $\mathbf{y}|_I = F(q)$,*
- *$\mathbf{x}(i), \mathbf{y}(i) \geq mr^3(3drm)^d$ for every $i \notin I$, and*
- *$\mathbf{y} - \mathbf{x}$ is the sum of the displacement of a path from p to q and a vector in the subgroup of $(\mathbb{Z}^d, +)$ generated by the displacements of the cycles.*

Then $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ for a path π such that $|\pi| \leq (\|\mathbf{y} - \mathbf{x}\| + drm)r^3(3drm)^{2d}$.

In this section, we fix such a structurally reversible PNS G. Since G is a disjoint union of strongly connected components, we can assume without loss of generality that G is strongly connected. The proof of Lemma 5 follows an extended form of the *zigzag-freeness* approach introduced in [14]. Intuitively, we fix an elementary path π_0 from p to q, and we prove that there exists a sequence $\theta_1, \ldots, \theta_k$ of short cycles on q such that for every $n \in \{0, \ldots, k\}$ the displacement of $\Delta(\theta_1 \ldots \theta_n)$ is almost the vector $\frac{n-d}{k}\mathbf{z}$ where $\mathbf{z} \overset{\text{def}}{=} \mathbf{y} - \mathbf{x} - \Delta(\pi_0)$. This result is based on the following lemma.

▶ **Lemma 6** ([9]). *Let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be a non-empty sequence of vectors in $\mathbb{R}^d$ such that $\|\mathbf{v}_j\|_\infty \leq 1$ for every $1 \leq j \leq k$ and let $\mathbf{v} = \sum_{j=1}^k \mathbf{v}_j$. There exists a permutation σ of $\{1, \ldots, k\}$ such that for every $n \in \{d, \ldots, k\}$, we have:*

$$\|\sum_{j=1}^{n} \mathbf{v}_{\sigma(j)} - \frac{n-d}{k}\mathbf{v}\|_\infty \leq d$$

From the previous lemma we deduce the following two corollaries.

▶ **Corollary 7.** *Let $\mathbf{Z}$ be a set of vectors in $\{-m, \ldots, m\}^d$ for some positive natural number m, and assume that $\mathbf{z}$ is a finite sum of vectors in $\mathbf{Z}$. Then $\mathbf{z}$ is a finite sum of at most $(\|\mathbf{z}\| + 1)(3dm)^d$ vectors in $\mathbf{Z}$.*

Proof. By symmetry, we can assume without loss of generality that $\mathbf{z} \geq \mathbf{0}$. Let k be the minimal natural number such that there exists a sequence $\mathbf{z}_1, \ldots, \mathbf{z}_k$ of vectors in $\mathbf{Z}$ such that $\mathbf{z} = \mathbf{z}_1 + \cdots + \mathbf{z}_k$. If $k = 0$ the lemma is proved, so let us assume that $k \geq 1$. Observe that there exists a sequence $\mathbf{e}_1, \ldots, \mathbf{e}_k$ of vectors in $\mathbb{N}^d$ such that $\mathbf{z} = \sum_{j=1}^k \mathbf{e}_j$ and such that $\mathbf{e}_j(i) \leq \max\{0, \mathbf{z}_j(i)\}$ for every $1 \leq i \leq d$ and every $1 \leq j \leq k$. We introduce the sequence $\mathbf{v}_1, \ldots, \mathbf{v}_k$ defined by $\mathbf{v}_j \overset{\text{def}}{=} \mathbf{z}_j - \mathbf{e}_j$. Notice that $\|\mathbf{v}_j\|_\infty \leq m$ and $\sum_{j=1}^k \mathbf{v}_j = \mathbf{0}$. We introduce $\mathbf{x}_n \overset{\text{def}}{=} \sum_{j=1}^n \mathbf{v}_j$. By applying a permutation, Lemma 6 applied on the sequence $(\frac{1}{m}\mathbf{v}_j)_{1 \leq j \leq n}$ shows that we can assume without loss of generality that $\mathbf{x}_n \in \mathbf{X}$ for every $d \leq n \leq k$ where $\mathbf{X}$ is the set of vectors $\mathbf{x} \in \mathbb{Z}^d$ such that $\|\mathbf{x}\|_\infty \leq md$. Notice that if $n \in \{0, \ldots, d\}$, we also have $\mathbf{x}_n \in \mathbf{X}$ since $\mathbf{x}_n$ is a sum of at most d vectors with a nom bounded by m.

The cardinal of $\mathbf{X}$ is bounded by $(1+2dm)^d \leq (3dm)^d$. Now, assume by contradiction that there exists $\ell \in \{0, \dots, k-(3dm)^d\}$ satisfying $\mathbf{e}_j = \mathbf{0}$ for every $j \in \{\ell+1, \dots, \ell+(3dm)^d\}$. Notice that there exists $p < q$ in $\{\ell, \dots, \ell+(3dm)^d\}$ such that $\mathbf{x}_p = \mathbf{x}_q$ since the cardinal of $\mathbf{X}$ is bounded by $(3dm)^d$. It follows that $\sum_{j=p+1}^{q} \mathbf{v}_j = \mathbf{0}$. From $\mathbf{e}_j = \mathbf{0}$ for every $j \in \{\ell+1, \dots, \ell+(3dm)^d\}$ it follows that $\mathbf{v}_j = \mathbf{z}_j$ for every $j \in \{p+1, \dots, q\}$. In particular $\sum_{j=p+1}^{q} \mathbf{z}_j = \mathbf{0}$. Hence k is not minimal since we can remove the vectors $\mathbf{z}_{p+1}, \dots, \mathbf{z}_q$ from the sequence $\mathbf{z}_1, \dots, \mathbf{z}_k$, and we get a contradiction. It follows that for every $\ell \in \{0, \dots, k-(3dm)^d\}$ there exists $j \in \{\ell+1, \dots, \ell+(3dm)^d\}$ such that $\mathbf{e}_j \neq \mathbf{0}$. From $\|\mathbf{z}\| = \sum_{j=1}^{k} \|\mathbf{e}_j\|$, it follows that $\|\mathbf{z}\| \geq \frac{k}{(3dm)^d} - 1$. Hence $k \leq (\|\mathbf{z}\|+1)(3dm)^d$. ◀

▶ Remark 8. The bound $(\|\mathbf{z}\|+1)(3dm)^d$ provided by the previous lemma is better than the bound $(\|\mathbf{z}\|+1)(2+(3m+1)^d)^d$ that one can derive from [15] by introducing the linear system $\mathbf{z} = \sum_{j=1}^{k} n_j \mathbf{z}_j$ over the free variables $\mathbf{z}, n_1, \dots, n_k$, where k is the cardinal of $\mathbf{Z}$, and $\{\mathbf{z}_1, \dots, \mathbf{z}_k\} = \mathbf{Z}$.

▶ **Corollary 9.** *Assume that $\mathbf{z} = \mathbf{z}_1 + \cdots + \mathbf{z}_k$ where $\mathbf{z}_1, \dots, \mathbf{z}_k$ are vectors in $\{-m, \dots, m\}^d$ for some positive natural number $m \geq 1$. There exists a permutation σ of $\{1, \dots, k\}$ such that for every $n \in \{0, \dots, k\}$ and for every $i \in \{1, \dots, d\}$, we have:*

$$\sum_{j=1}^{n} \mathbf{z}_{\sigma(j)}(i) \geq \min\{\mathbf{z}(i), 0\} - md$$

Proof. If $k = 0$ the lemma is proved. So, we can assume that $k \geq 1$. By applying a permutation, Lemma 6 on the sequence $(\frac{1}{m}\mathbf{z}_j)_{1 \leq j \leq k}$ shows that we can assume without loss of generality that for every $n \in \{0, \dots, k\}$, there exists a vector $\mathbf{e}_n \in \mathbb{R}^d$ such that $\|\mathbf{e}_n\|_\infty \leq md$ and such that $\mathbf{x}_n = \frac{n-d}{k}\mathbf{z} + \mathbf{e}_n$ where $\mathbf{x}_n \overset{\text{def}}{=} \sum_{j=1}^{n} \mathbf{z}_j$. Let $i \in \{1, \dots, d\}$ and let us prove that $\mathbf{x}_n(i) \geq \min\{\mathbf{z}(i), 0\} - md$. Observe that if $n \in \{0, \dots, d\}$ then the property is immediate since $\mathbf{x}_n(i) \geq -md$. So, let us assume that $n > d$. If $\mathbf{z}(i) \geq 0$ then $\frac{n-d}{k}\mathbf{z}(i) \geq 0$ and we get $\mathbf{x}_n(i) \geq \mathbf{e}_n(i) \geq -md$. If $\mathbf{z}(i) \leq 0$ then $\frac{n-d}{k}\mathbf{z}(i) \geq \mathbf{z}(i)$. In particular $\mathbf{x}_n(i) \geq \min\{\mathbf{z}(i), 0\} - md$ also in that case. ◀

A cycle is said to be *full-state* if every state occurs in the cycle. We first prove that there exists a "short" full-state cycle with a zero displacement thanks to the following lemma.

▶ **Lemma 10.** *Every transition occurs on a finite sequence $\theta_1, \dots, \theta_n$ of (eventually disjoint) simple cycles such that $\Delta(\theta_1) + \cdots + \Delta(\theta_n) = \mathbf{0}$ and such that $n \leq (3drm)^d$.*

Proof. Let t be a transition. Since G is strongly connected, the transition t occurs in a simple cycle θ_0. Lemma 4 shows that $-\Delta(\theta_0)$ is a finite sum of displacements of simple cycles. In particular $-\Delta(\theta_0)$ is in the cone generated by the displacements of simple cycles, i.e. the finite sums of displacements of simple cycles multiplied by non-negative rational numbers. From Carathéodory theorem, there exists d simple cycles $r_1, \dots, r_d$ and d non-negative rational numbers $\lambda_1, \dots, \lambda_d$ such that $-\Delta(\theta_0) = \sum_{j=1}^{d} r_j \Delta(\theta_j)$. By introducing a positive integer β_0 such that $\beta_j \overset{\text{def}}{=} \beta_0 r_j$ is a natural number for every j, we derive that the following linear system over the sequences $(\beta_j)_{0 \leq\ j \leq d}$ of natural numbers

$$\sum_{j=0}^{d} \beta_j \mathbf{v}_j = \mathbf{0}$$

admits a solution satisfying $\beta_0 > 0$ where $\mathbf{v}_j \overset{\text{def}}{=} \Delta(\theta_j)$.

From [15], it follows that solutions of that system can be decomposed as finite sums of "minimal" solutions $(\beta_j)_{1\leq j\leq k}$ of the same system satisfying additionally the following constraint:

$$\sum_{j=0}^{d} \beta_j \leq (1+(d+1)rm)^d$$

From $1+(d+1)rm \leq (3drm)$, we derive $(1+(d+1)rm)^d \leq (3drm)^d$. Since there exist solutions of that system with $\beta_0 > 0$, there exists at least a minimal one satisfying the same constraint. We have proved the lemma. ◀

▶ **Lemma 11.** *There exists a full-state cycle with a zero displacement with a length bounded by* $r^2(r-1)(3drm)^d$.

Proof. Let us consider the set H of pairs $(p,q) \in Q \times Q$ such that there exists a transition from p to q with $p \neq q$. For every $h \in H$ of the form (p,q), we select a transition $t_h \in T$ from p to q. Lemma 10 shows that for every $h \in H$, there exists a sequence of at most $(3drm)^d$ simple cycles with a zero total displacement. It follows that there exists a sequence of at most $|H|(3drm)^d$ simple cycles with a zero total displacement that contains all the transitions t_h with $h \in H$. Since the set of transitions that occurs in that sequence is strongly connected, Euler's Lemma shows that there exists a cycle θ with the same Parikh image as the sum of the Parikh images of the cycles occurring in the sequence. It follows that $|\theta| \leq r|H|(3rdm)^d$. Notice that $\Delta(\theta) = \mathbf{0}$ and θ is a full-state cycle. From $|H| \leq r(r-1)$ we are done. ◀

Now, let us prove Lemma 5. Let π_0 be an elementary path from p to q, and let $\mathbf{z} \overset{\text{def}}{=} \mathbf{y} - \mathbf{x} - \Delta(\pi_0)$.

Let us first explain why $\mathbf{z}$ is a finite sum of displacements of simple cycles. By hypothesis, there exists a path π_1 from p to q such that $\mathbf{y} - \mathbf{x} - \Delta(\pi_1)$ is in the group generated by displacements of cycles. Let π' be a path from q to p and observe that $\mathbf{z} = (\mathbf{y} - \mathbf{x} - \Delta(\pi_1)) - \Delta(\pi'\pi_0) + \Delta(\pi'\pi_1)$. Since $\pi'\pi_0$ and $\pi'\pi_1$ are two cycles, it follows that $\mathbf{z}$ is in the group generated by the displacements of the cycles. Lemma 4 shows that $\mathbf{z}$ is finite sum of displacements of simple cycles.

Corollary 7 and Corollary 9 shows that there exists a sequence $\mathbf{z}_1, \ldots, \mathbf{z}_k$ of displacements of simple cycles such that $\mathbf{z} = \sum_{j=1}^{k} \mathbf{z}_j$, $k \leq (1+\|\mathbf{z}\|)(3drm)^d$, and such that for every $n \in \{0, \ldots, k\}$, we have:

$$\sum_{j=1}^{n} \mathbf{z}_j(i) \geq \min\{0, \mathbf{z}(i)\} - drm$$

Lemma 11 shows that there exists a full-state cycle θ_0 with a zero displacement with a length bounded by $r^2(r-1)(3drm)^d$. Thanks to a rotation of θ_0, we can assume that θ_0 is a cycle on q. Now, observe that for every $1 \leq j \leq k$, there exists a simple cycle θ_j' with a displacement equal to $\mathbf{z}_j$. By inserting θ_j' in the full-state cycle θ_0, we get a cycle θ_j on q. Notice that $\Delta(\theta_j) = \mathbf{z}_j$ and $|\theta_j| \leq r^2(r-1)(3drm)^d + r$. We introduce the path π defined as follows:

$$\pi \overset{\text{def}}{=} \pi_0\theta_1\ldots\theta_n$$

We are going to prove that $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$. To do so, let u be a prefix of π and let $i \in \{1, \ldots, d\}$ and let us prove that $\mathbf{x}(i) + \Delta(u)(i) \geq 0$. Oberve that if $i \in I$, the flow function F shows that $\mathbf{x}(i) + \Delta(u)(i) = F(q_u)(i) \geq 0$ where q_u is a state reached from p by reading u. So, we can assume that $i \notin I$. Observe that if u is a prefix of π_0 the property is immediate since $\Delta(u)(i) \geq -m|u| \geq -mr$. In particular $\mathbf{x}(i)+\Delta(u)(i) \geq 0$. So, we can assume that there exists $n \in \{1,\ldots,k\}$ and a prefix u' of θ_n such that $u = \pi_0\theta_1\ldots\theta_{n-1}u'$. It follows that $\Delta(u) = \Delta(\pi_0)+\Delta(u')+\sum_{j=1}^{n-1}\mathbf{z}_j(i)$. Moreover, notice that $\Delta(u')(i) \geq -m|u'| \geq -mr^2(r-1)(3drm)^d - mr$ for every $i \in \{1,\ldots,d\}$.

We decompose the proof that $\mathbf{x}(i)+\Delta(u)(i) \geq 0$ in two cases following that $\mathbf{z}(i) \leq 0$ or $\mathbf{z}(i) \geq 0$.

- Assume first that $\mathbf{z}(i) \geq 0$. In that case $\sum_{j=1}^{n-1}\mathbf{z}_j(i) \geq -drm$. It follows that $\Delta(u)(i) \geq -mr - mr^2(r-1)(3drm)^d - mr - drm \geq -mr^3(3drm)^d$. Hence $\mathbf{x}(i)+\Delta(u)(i) \geq 0$.
- Now, assume that $\mathbf{z}(i) \leq 0$. In that case $\sum_{j=1}^{n-1}\mathbf{z}_j(i) \geq \mathbf{z}(i) - drm$. It follows that $\mathbf{x}(i)+\Delta(u)(i) \geq \mathbf{x}(i)+\Delta(\pi_0)+\mathbf{z}(i)+\Delta(u')(i) - drm = \mathbf{y}(i) - \Delta(u')(i) - drm \geq \mathbf{y}(i) - mr^2(r-1)(3drm)^d - mr \geq 0$.

We have proved that $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$. Now, observe that $|\pi| \leq r+k(r^2(r-1)(3drm)^d+r)$. From $k \leq (1+\|\mathbf{z}\|)(3drm)^d$ and $\|\mathbf{z}\| \leq \|\mathbf{y}-\mathbf{x}\|+d(r-1)m$, we get $|\pi| \leq (\|\mathbf{y}-\mathbf{x}\|+drm)r^3(3drm)^{2d}$. Lemma 5 is proved.

4 Extractors

The notion of extractors was first introduced in [11]. Intuitively, extractors provides a natural way to classify components of a vector of natural numbers into two categories: large ones and small ones. The notion is parametrized by a set $I \subseteq \{1,\ldots,d\}$ that provides a way to focus only on components in I. More formally, a *d-dimensional extractor* λ is a non-decreasing sequence $(\lambda_0 \leq \cdots \leq \lambda_{d+1})$ of positive natural numbers denoting some *thresholds*. Given a d-dimensional extractor λ and a set $I \subseteq \{1,\ldots,d\}$, a *$(\lambda,I)$-small set* of a set $\mathbf{C} \subseteq \mathbb{N}^d$ is a subset $J \subseteq I$ such that $\mathbf{c}(j) < \lambda_{|J|}$ for every $j \in J$ and $\mathbf{c} \in \mathbf{C}$. The following lemma shows that there exists a unique maximal (λ,I)-small set w.r.t. inclusion. We denote by $\text{extract}_{\lambda,\mathbf{C}}(I)$ this set.

▶ **Lemma 12.** *The class of (λ,I)-small sets of a set $\mathbf{C} \subseteq \mathbb{N}^d$ is non empty and stable under union.*

Proof. We adapt the proof of [11, Section 8]. Since the class contains the empty set, it is nonempty. Now, let us prove the stability by union by considering two (λ,I)-small sets J_1 and J_2 of $\mathbf{C}$ and let us prove that $J \overset{\text{def}}{=} J_1 \cup J_2$ is a (λ,I)-small set of $\mathbf{C}$. Since $J_1,J_2 \subseteq I$, we derive $J \subseteq I$. Let $\mathbf{c} \in \mathbf{C}$ and $j \in J$. If $j \in J_1$ then $\mathbf{c}(j) < \lambda_{|J_1|} \leq \lambda_{|J|}$ since $|J_1| \leq |J|$. Symmetrically, if $j \in J_2$ we deduce that $\mathbf{c}(j) < \lambda_{|J_2|} \leq \lambda_{|J|}$. We have proved that J is a (λ,I)-small set of $\mathbf{C}$. ◀

▶ **Example 13.** Let us consider the 2-dimensional extractor $\lambda = (\lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \lambda_3)$ and assume that $I = \{1,2\}$ and let $\mathbf{C} = \{(m,n)\}$ with $m,n \in \mathbb{N}$. We have:

$$\text{extract}_{\lambda,X}(I) = \begin{cases} \{1,2\} & \text{if } m,n < \lambda_2 \\ \emptyset & \text{if } (m \geq \lambda_2 \wedge n \geq \lambda_1) \vee (m \geq \lambda_1 \wedge n \geq \lambda_2) \\ \{1\} & \text{if } m < \lambda_1 \wedge n \geq \lambda_2 \\ \{2\} & \text{if } m \geq \lambda_2 \wedge n < \lambda_1 \end{cases}$$

▶ Remark 14. As shown by the previous example, the values λ_0 and λ_{d+1} of any d-dimensional extractor λ are not used directly by our definitions. Those extremal values are introduced to simplify some notations in the sequel.

The following lemma shows that components that are not in $\text{extract}_{\lambda,\mathbf{C}}(I)$ are large for at least one vector in $\mathbf{C}$.

▶ **Lemma 15.** *Let* $J \overset{\text{def}}{=} \text{extract}_{\lambda,\mathbf{C}}(I)$*. For every* $i \in I \backslash J$ *there exists* $\mathbf{c} \in \mathbf{C}$ *such that:*

$$\mathbf{c}(i) \geq \lambda_{|J|+1}$$

Proof. Assume that for some $i \in I \backslash J$, we have $\mathbf{c}(i) < \lambda_{|J|+1}$ for every $\mathbf{c} \in \mathbf{C}$. Let $J' \overset{\text{def}}{=} J \cup \{i\}$ and observe that J' is a (λ, I)-small set of $\mathbf{C}$. In fact, for every $\mathbf{c} \in \mathbf{C}$ and for every $j \in J'$, we have $\mathbf{c}(j) < \lambda_{|J|} \leq \lambda_{|J'|}$ if $j \in J$, and $\mathbf{c}(j) < \lambda_{|J|+1} = \lambda_{|J'|}$ if $j = i$. We get a contradiction by maximality of $\text{extract}_{\lambda,\mathbf{C}}(I)$. We deduce the lemma. ◀

Given a set $I \subseteq \{1, \ldots, d\}$ we define $\text{extract}_{\lambda,e}(I)$ for a finite word e of configurations by $\text{extract}_{\lambda,\varepsilon}(I) \overset{\text{def}}{=} I$, and by $\text{extract}_{\lambda,e\mathbf{c}}(I) \overset{\text{def}}{=} \text{extract}_{\lambda,\{\mathbf{c}\}}(\text{extract}_{\lambda,e}(I))$ for every $\mathbf{c} \in \mathbb{N}^d$ and for every finite word e of configurations. Given an infinite word e of configurations, we observe that $(\text{extract}_{\lambda,e_n}(I))_{n \in \mathbb{N}}$ where e_n is the finite prefix of e of length n is a non-increasing sequence of sets in $\{1, \ldots, d\}$. It follows that this sequence is asymptotically constant and equals to a set included in $\{1, \ldots, d\}$. We denote $\text{extract}_{\lambda,e}(I)$ that set. The following lemma shows that extracting along a word of configurations in $\mathbf{C}$ asymptotically coincides with an extraction of $\mathbf{C}$.

▶ **Lemma 16.** *Let us consider a set* $I \subseteq \{1, \ldots, d\}$*, an extractor* λ*, a set* $\mathbf{C}$ *of configurations, and an infinite word* e *over* $\mathbf{C}$*. We have* $\text{extract}_{\lambda,\mathbf{C}}(I) \subseteq \text{extract}_{\lambda,e}(I)$*. Moreover,* $\text{extract}_{\lambda,\mathbf{C}}(I) = \text{extract}_{\lambda,e}(I)$ *if every configuration of* $\mathbf{C}$ *occurs infinitely often in* e*.*

Proof. We introduce $J \overset{\text{def}}{=} \text{extract}_{\lambda,\mathbf{C}}(I)$, $J_\infty \overset{\text{def}}{=} \text{extract}_{\lambda,e}(I)$, the prefix e_n of length n of e, and $J_n \overset{\text{def}}{=} \text{extract}_{\lambda,e_n}(I)$.

Let us prove that $J \subseteq J_n$ for every n. Since $J_0 = I$ the property is proved for $n = 0$. Assume that $J \subseteq J_{n-1}$ for some $n \geq 1$ and let us prove that $J \subseteq J_n$. There exists $\mathbf{c} \in \mathbf{C}$ such that $e_n = e_{n-1}\mathbf{c}$. Since $\mathbf{c} \in \mathbf{C}$, it follows that $\mathbf{c}(j) < \lambda_{|J|}$ for every $j \in J$. As $J \subseteq J_{n-1}$, we deduce that J is a (λ, J_{n-1})-small set of $\{\mathbf{c}\}$. Since J_n is the maximal set satisfying that property, we get $J \subseteq J_n$ and we have proved the induction. It follows that $J \subseteq J_n$ for every $n \in \mathbb{N}$. Moreover, since $J_\infty = \bigcap_{n \in \mathbb{N}} J_n$, we deduce the inclusion $J \subseteq J_\infty$.

Now, assume that every $\mathbf{c} \in \mathbf{C}$ occurs in e infinitely often. Since $(J_n)_{n \in \mathbb{N}}$ is a non increasing sequence of $\{1, \ldots, d\}$, there exists N such that $J_n = J_\infty$ for every $n \geq N$. Let $\mathbf{c} \in \mathbf{C}$. There exists $n > N$ such that $e_n = e_{n-1}\mathbf{c}$. From $J_n = \text{extract}_{\lambda,\{\mathbf{c}\}}(J_{n-1})$ and $J_n = J_{n-1} = J_\infty$, we derive $J_\infty = \text{extract}_{\lambda,\{\mathbf{c}\}}(J_\infty)$. In particular $\mathbf{c}(j) < \lambda_{|J_\infty|}$ for every $j \in J_\infty$. We have proved that $\mathbf{c}(j) < \lambda_{|J_\infty|}$ for every $j \in J_\infty$ and for every $\mathbf{c} \in \mathbf{C}$. As $J_\infty \subseteq I$, we deduce that J_∞ is a (λ, I)-small set of $\mathbf{C}$. Since J is the maximal set satisfying that property, we deduce that $J_\infty \subseteq J$. It follows that $J = J_\infty$. ◀

5 Rackoff Extraction

An A^*-execution e is said to be I*-cyclic* for some $I \subseteq \{1, \ldots, d\}$ if $\text{src}(e)|_I = \text{tgt}(e)|_I$. We say that a word $\sigma = \mathbf{a}_1 \ldots \mathbf{a}_k$ of actions in A is obtained from an A^*-execution e by removing I-cycles where $I \subseteq \{1, \ldots, d\}$, if there exists a decomposition of e into a concatenation $e_0 \ldots e_k$ of I-cyclic A^*-executions $e_0, \ldots, e_k$ such that $\text{tgt}(e_{j-1}) \xrightarrow{a_j} \text{src}(e_j)$ for every $1 \leq j \leq k$.

An extractor $\lambda = (\lambda_0 \leq \cdots \leq \lambda_{d+1})$ is said to be m*-adapted* if for every $n \in \{0, \ldots, d\}$:

$$\lambda_{n+1} \geq \lambda_n + m\lambda_n^n$$

▶ **Lemma 17** (slight extension of [16]). *Let λ be an m-adapted extractor and e be an A^*-execution for a PN $A \subseteq \{0,\dots,m\}^d \times \mathbb{N}^d$. Let $I \overset{\text{def}}{=} \text{extract}_{\lambda,e}(\{1,\dots,d\})$. There exists a word σ that can be obtained from e by removing I-cycles such that*

$$|\sigma| \leq \sum_{j=1}^{d} \lambda_j^j$$

and such that $\text{src}(e) \xrightarrow{\sigma} \mathbf{c}$ *for some configuration* $\mathbf{c}$ *satisfying* $\mathbf{c}(i) = \text{tgt}(e)(i)$ *for every $i \in I$, and such that for every $i \notin I$ we have:*

$$\mathbf{c}(i) \geq \lambda_{|I|+1} - m \sum_{j=1}^{|I|} \lambda_j^j$$

Proof. The proof follows a similar approach to the original one from Rackoff [16]. A detailed proof is given in a long version of the paper available online. ◀

6 Strongly-Connected Components of Configurations

A *strongly-connected component of configurations* of a PN A (*SCCC* for short) is a strongly-connected component of the reachability graph $(\mathbb{N}^d, \xrightarrow{A})$.

We associate to an extractor λ and a SCCC $\mathbf{C}$ of a PN A, a PNS G defined as follows. We introduce the set $I \overset{\text{def}}{=} \text{extract}_{\lambda,\mathbf{C}}(\{1,\dots,d\})$, the set of states $Q \overset{\text{def}}{=} \{\mathbf{c}|_I \mid \mathbf{c} \in \mathbf{C}\}$ and the set of transitions $T \overset{\text{def}}{=} \{(\mathbf{x}|_I, a, \mathbf{y}|_I) \mid (\mathbf{x}, a, \mathbf{y}) \in \mathbf{C} \times A \times \mathbf{C} \wedge \mathbf{x} \xrightarrow{a} \mathbf{y}\}$. Notice that Q is finite since it contains at most $\lambda_{|I|}^{|I|}$ elements. In particular T is finite as well. The PNS G is defined as the tuple $\langle Q, A, T\rangle$.

▶ **Lemma 18.** *The PNS G is structurally reversible.*

Proof. Let (p, a, q) be a transition in T. There exist $\mathbf{x}, \mathbf{y} \in \mathbf{C}$ such that $\mathbf{x} \xrightarrow{a} \mathbf{y}$ and such that $p = \mathbf{x}|_I$ and $q = \mathbf{y}|_I$. Moreover since $\mathbf{C}$ is a SCCC, there exists a word σ of actions in A such that $\mathbf{y} \xrightarrow{\sigma} \mathbf{x}$. We deduce that there exists a path in G from q to p labeled by σ. Notice that $\Delta(a) + \Delta(\sigma) = \mathbf{y} - \mathbf{x} + \mathbf{x} - \mathbf{y} = \mathbf{0}$. It follows that G is structurally reversible. ◀

Let us prove the following technical lemma.

▶ **Lemma 19.** *If $\mathbf{C}$ is not reduced to a singleton, there exists an A^ω-execution e of configurations in $\mathbf{C}$ such that every configuration of $\mathbf{C}$ occurs infinitely often in e.*

Proof. Since $\mathbf{C}$ is countable, there exists an infinite sequence $(\mathbf{c}_n)_{n\in\mathbb{N}}$ such that $\mathbf{C} = \{\mathbf{c}_n \mid n \in \mathbb{N}\}$. Moreover, by replacing that sequence by the sequence $s_0, s_1, \dots$ where $s_n \overset{\text{def}}{=} \mathbf{c}_0, \dots, \mathbf{c}_n$, we can assume without loss of generality that every configuration of $\mathbf{C}$ occurs infinitely often in the sequence $(\mathbf{c}_n)_{n\in\mathbb{N}}$. Since $\mathbf{C}$ is a SCCC, for every positive natural number n, there exists an A^*-execution from $\mathbf{c}_{n-1}$ to $\mathbf{c}_n$ of the form $e_n\mathbf{c}_n$. Let us introduce the word $e \overset{\text{def}}{=} e_1 e_2 \dots$. Notice that since $\mathbf{C}$ is not reduced to a singleton, the word e is infinite. Moreover, notice that e is an A^ω-execution satisfying the lemma. ◀

Now, assume that λ is m-adapted for some positive natural number m.

▶ **Lemma 20.** *If $A \subseteq \{0, \dots, m\}^d \times \mathbb{N}^d$, for every $\mathbf{x} \in \mathbf{C}$, there exists a cycle in G on $\mathbf{x}|_I$ labeled by a word u such that:*

$$|u| \leq \sum_{j=1}^{d} \lambda_j^j$$

and a configuration $\mathbf{x}'$ such that $\mathbf{x} \xrightarrow{u} \mathbf{x}'$, $\mathbf{x}'|_I = \mathbf{x}|_I$ and such that $\mathbf{x}'(i) \geq \lambda_{|I|+1} - m \sum_{j=1}^{|I|} \lambda_j^j$ for every $i \notin I$.

Proof. Observe that if $\mathbf{C}$ is reduced to a singleton, the lemma is trivial with $u \overset{\text{def}}{=} \varepsilon$. So, we can assume that $\mathbf{C}$ is not a singleton. Lemma 19 shows that there exists an A^ω-execution $e = \mathbf{c}_0\mathbf{c}_1 \dots$ of configurations in $\mathbf{C}$ such that every configuration of $\mathbf{C}$ occurs infinitely often. Without loss of generality, by replacing e by a suffix of e we can assume that $\mathbf{x} = \mathbf{c}_0$. Lemma 16 shows that $\text{extract}_{\lambda,e}(\{1, \dots, d\}) = I$. It follows that there exists $N \in \mathbb{N}$ such that for every $n \geq N$ the prefix e_n of e of length n satisfies $\text{extract}_{\lambda,e_n}(\{1, \dots, d\}) = I$. Since $\mathbf{x}$ occurs infinitely often in e, there exists $n \geq N$ such that $\mathbf{x}$ is the last configuration of e_n. Lemma 17 shows that there exists a word u that can be obtained from e_n by removing I-cycles such that

$$|u| \leq \sum_{j=1}^{d} \lambda_j^j$$

and such that $\mathbf{x} \xrightarrow{u} \mathbf{x}'$ for some configuration $\mathbf{x}'$ satisfying $\mathbf{x}'|_I = \mathbf{x}|_I$, and such that for every $i \notin I$ we have:

$$\mathbf{x}'(i) \geq \lambda_{|I|+1} - m \sum_{j=1}^{|I|} \lambda_j^j$$

Since u can be obtained from e_n by removing I-cycles, it follows that u is the label of a cycle on $\mathbf{x}|_I$ in the PNS G. ◀

Symmetrically, we deduce a similar backward property.

▶ **Lemma 21.** *If $A \subseteq \mathbb{N}^d \times \{0, \dots, m\}^d$, for every $\mathbf{y} \in \mathbf{C}$, there exists a cycle in G on $\mathbf{y}|_I$ labeled by a word v such that:*

$$|v| \leq \sum_{j=1}^{d} \lambda_j^j$$

and a configuration $\mathbf{y}'$ such that $\mathbf{y}' \xrightarrow{v} \mathbf{y}$, $\mathbf{y}'|_I = \mathbf{y}|_I$, and such that for every $i \notin I$: $\mathbf{y}'(i) \geq \lambda_{|I|+1} - m \sum_{j=1}^{|I|} \lambda_j^j$.

Proof. Let us introduce the PN $A' \overset{\text{def}}{=} \{(\mathbf{a}_+, \mathbf{a}_-) \mid (\mathbf{a}_-, \mathbf{a}_+) \in A\}$. Observe that $\mathbf{C}$ is a SCCC of A'. Let G' be the PNS associated to the extractor λ and the SCCC $\mathbf{C}$ of A'. Lemma 22 shows that there exists a cycle in G' on $\mathbf{y}|_I$ labeled by a word u such that:

$$|u| \leq \sum_{j=1}^{d} \lambda_j^j$$

and a configuration $\mathbf{y}'$ such that $\mathbf{y} \xrightarrow{u} \mathbf{y}'$, $\mathbf{y}|_I = \mathbf{y}'|_I$, and such that $\mathbf{y}'(i) \geq \lambda_{|I|+1} - m \sum_{j=1}^{|I|} \lambda_j^j$ for every $i \notin I$. Assume that $u = a_1' \dots a_n'$ with $a_j' = (\mathbf{x}_j, \mathbf{y}_j)$ and let $v \overset{\text{def}}{=} a_1 \dots a_n$ with $a_j \overset{\text{def}}{=} (\mathbf{y}_j, \mathbf{x}_j)$. Observe that since u is a cycle on $\mathbf{y}|_I$ in G', then v is a cycle on $\mathbf{y}|_I$ in G. Moreover, from $\mathbf{y} \xrightarrow{u} \mathbf{y}'$ we derive $\mathbf{y}' \xrightarrow{v} \mathbf{y}$. We have proved the lemma. ◀

7 Mutually Reachable Configurations

In this section, we prove Theorem 2. We consider a PN $A \subseteq \{0, \dots, m\}^d \times \{0, \dots, m\}^d$ for some positive natural number m. We consider two mutually reachable configurations $\mathbf{x}, \mathbf{y}$ for A. Since the theorem is trivial when $\mathbf{x} = \mathbf{y}$, we can assume that $\mathbf{x} \neq \mathbf{y}$. In particular $\|\mathbf{y} - \mathbf{x}\| \geq 1$.

We let $\mathbf{C}$ be the SCCC of A containing $\mathbf{x}$ and $\mathbf{y}$. We introduce the extractor λ satisfying $\lambda_0 = 1$, and for every $n \in \{0, \dots, d\}$:

$$\lambda_{n+1} \overset{\text{def}}{=} m \sum_{j=1}^{n} \lambda_j^j + m\lambda_n^{3n}(3d\lambda_n^n m)^d$$

Observe that λ is m-adapted. We introduce $I \overset{\text{def}}{=} \text{extract}_{\lambda,\mathbf{C}}(\{1, \dots, d\})$ and the structurally reversible PNS G associated to $\mathbf{C}$, λ and $\mathbf{A}$. Notice that the number of states of G is bounded by $r \overset{\text{def}}{=} \lambda_{|I|}^{|I|}$. We introduce the states p, q of G defined as $p \overset{\text{def}}{=} \mathbf{x}|_I$ and $q \overset{\text{def}}{=} \mathbf{y}|_I$. Observe that $\mathbf{y} - \mathbf{x}$ is the displacement of a path from p to q in G. We introduce the flow function $F : Q \to \mathbb{N}^I$ defined as the identity.

Let us observe that $\lambda_j \leq \lambda_d$ for every $j \in \{1, \dots, d\}$. In particular $r \leq \lambda_d^d$.

▶ **Lemma 22.** *The PNS G admits a cycle on p labeled by a word u and a cycle on q labeled by a word v such that:*

$$|u|, |v| \leq d\lambda_d^d$$

and such that there exist configurations $\mathbf{x}', \mathbf{y}'$ such that $\mathbf{x} \xrightarrow{u} \mathbf{x}'$, $\mathbf{y}' \xrightarrow{v} \mathbf{y}$, and such that for every $i \notin I$, we have:

$$\mathbf{x}'(i), \mathbf{y}'(i) \geq mr^3(3drm)^d$$

Proof. This lemma is a direct corollary of Lemma 20 and Lemma 21. ◀

From $\mathbf{y}' - \mathbf{x}' = \mathbf{y} - \mathbf{x} - \Delta(u) - \Delta(v)$, we deduce from Lemma 5 that there exists a word σ of actions in $\mathbf{A}$ such that $\mathbf{x}' \xrightarrow{\sigma} \mathbf{y}'$ and such that $|\sigma| \leq (\|\mathbf{y}' - \mathbf{x}'\| + drm)r^3(3drm)^{2d}$. Observe that we have:

$$\begin{aligned}\|\mathbf{y}' - \mathbf{x}'\| &\leq \|\mathbf{y} - \mathbf{x}\| + \|\Delta(u)\| + \|\Delta(v)\| \\ &\leq \|\mathbf{y} - \mathbf{x}\| + dm(|u| + |v|) \\ &\leq \|\mathbf{y} - \mathbf{x}\| + 2d^2m\lambda_d^d\end{aligned}$$

Let $w = u\sigma v$. Observe that $\mathbf{x} \xrightarrow{w} \mathbf{y}$. We derive:

$$\begin{aligned}|w| &\leq 2d\lambda_d^d + (\|\mathbf{y} - \mathbf{x}\| + 2d\lambda_d^d m + d\lambda_d^d m)\lambda_d^{3d}(3d\lambda_d^d m)^{2d} \\ &\leq \|\mathbf{y} - \mathbf{x}\| 8d\lambda_d^d m\lambda_d^{3d}(3d\lambda_d^d m)^{2d} \\ &\leq \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|(3d\lambda_d^d m)^{6d}\end{aligned}$$

From the following Lemma 23 we derive:

$$|w| \leq \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|(3dm)^{(d+1)^{2d+4}}$$

We deduce Theorem 2.

▶ **Lemma 23.** *We have:*

$$(3d\lambda_d^d m)^{6d} \leq (3dm)^{(d+1)^{2d+4}}$$

Proof. Assume first that $d = 1$. In that case, the definiton of λ_{n+1} with $n = 0$ provides $\lambda_1 = 3m^2$ and the lemma is immediate. So, let us assume that $d \geq 2$. Notice that $\lambda_j^j \leq \lambda_n^n$ for every $j \in \{1, \dots, n\}$ for every $n \in \{0, \dots, d-1\}$. It follows that we have:

$$\begin{aligned}\lambda_{n+1} &\leq 2d\lambda_n^{3n} m(3d\lambda_n^n m)^d \\ &\leq (3d\lambda_n m)^{(d+1)^2-4}\end{aligned}$$

By induction, we deduce that for every $n \in \{0, \dots, d\}$, we have:

$$\lambda_n \leq (3dm)^{n((d+1)^2-4)^n}$$

In particular:

$$3d\lambda_d^d m \leq (3dm)^{d^2(d+1)^{2d}}$$

Hence

$$(3d\lambda_d^d m)^{6d} \leq (3dm)^{6d^3(d+1)^{2d}} \leq (3dm)^{(d+1)^{2d+4}}$$

where we use the inequality $6d^3 \leq (d+1)^4$. ◀

8 Conclusion

In this paper we proved that the distance in the reachability graph between two mutually reachable configurations is linear with respect to the Euclidean distance between those two configurations. As a future work, we would like to apply that result to provide lower bounds on the number of states of population protocols computing some predicates [3].

The author thanks Alain Finkel, Igor Khmelnitsky, and Serge Haddad for promoting him a Karp and Miller problem that motivated this work, and Matthias Englert and Ranko Lazic for pointing out the Steinitz constant lemma [9].

References

1 Gérard Basler, Michele Mazzucchi, Thomas Wahl, and Daniel Kroening. Symbolic Counter Abstraction for Concurrent Software. In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2009.

2 Eike Best and Javier Esparza. Existence of home states in Petri nets is decidable. *Inf. Process. Lett.*, 116(6):423–427, 2016.

3 Michael Blondin, Javier Esparza, and Stefan Jaax. Large Flocks of Small Birds: on the Minimal Size of Population Protocols. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. `doi:10.4230/LIPIcs.STACS.2018.16`.

4 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the Coverability Problem Continuously. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 480–496. Springer, 2016.

5 E. Cardoza, Richard J. Lipton, and Albert R. Meyer. Exponential Space Complete Problems for Petri Nets and Commutative Semigroups: Preliminary Report. In *STOC'76*, pages 50–54. ACM, 1976. doi:10.1145/800113.803630.

6 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The Reachability Problem for Petri Nets is Not Elementary (Extended Abstract). In *STOC*. ACM Computer Society, 2019. to appear.

7 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Inf.*, 54(2):191–215, 2017.

8 Thomas Geffroy, Jérôme Leroux, and Grégoire Sutre. Occam's Razor applied to the Petri net coverability problem. *Theor. Comput. Sci.*, 750:38–52, 2018.

9 V. S. Grinberg and S. V. Sevast'yanov. Value of the Steinitz constant. *Functional Analysis and Its Applications*, 14(2):125–126, April 1980. doi:10.1007/BF01086559.

10 Petr Jancar, Jérôme Leroux, and Grégoire Sutre. Co-finiteness and Co-emptiness of Reachability Sets in Vector Addition Systems with States. In *Petri Nets*, volume 10877 of *Lecture Notes in Computer Science*, pages 184–203. Springer, 2018.

11 Jérôme Leroux. Vector Addition System Reversible Reachability Problem. *Logical Methods in Computer Science*, 9(1), 2013.

12 Jérôme Leroux, M. Praveen, and Grégoire Sutre. A Relational Trace Logic for Vector Addition Systems with Application to Context-Freeness. In *CONCUR*, volume 8052 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2013.

13 Jérôme Leroux and Sylvain Schmitz. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *LICS*. IEEE Computer Society, 2019. to appear.

14 Jérôme Leroux and Grégoire Sutre. On Flatness for 2-Dimensional Vector Addition Systems with States. In *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2004.

15 Loic Pottier. Minimal Solutions of Linear Diophantine Systems: Bounds and Algorithms. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 1991.

16 C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

Boolean Algebras from Trace Automata

Alexandre Mansard
LIM, University of La Réunion, France
alexandre.mansard@univ-reunion.fr

Abstract

We consider trace automata. Their vertices are Mazurkiewicz traces and they accept finite words. Considering the length of a trace as the length of its Foata normal form, we define the operations of level-length synchronization and of superposition of trace automata. We show that if a family $\mathcal{F}$ of trace automata is closed under these operations, then for any deterministic automaton $H \in \mathcal{F}$, the word languages accepted by the deterministic automata of $\mathcal{F}$ that are length-reducible to H form a Boolean algebra. We show that the family of trace suffix automata with level-regular contexts and the subfamily of vector addition systems satisfy these closure properties. In particular, this yields various Boolean algebras of word languages accepted by deterministic vector addition systems.

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

Keywords and phrases Boolean algebras, traces, automata, synchronization, pushdown automata, vector addition systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.48

1 Introduction

In automatic verification, it is useful to highlight families of languages with good closure properties, as for example Boolean algebras of languages. For example, the fact that the first-order theory of any word-automatic graph[1] is decidable essentially relies on the Boolean closure properties of regular languages: to any relation defined by a first-order formula, there corresponds (in an effective way) a regular language and the problem of deciding whether the graph satisfies a given statement reduces to the problem of deciding emptiness for the corresponding regular language [13].

The regular, context-free, context-sensitive and recursively enumerable languages form a famous increasing hierarchy of formal language families defined by Chomsky in [8] from grammars of increasing complexity. It can also be obtained from families of automata. Indeed, regular languages are accepted by finite automata, context-free languages are accepted by pushdown automata (or more generally by word suffix automata [2]), context-sensitive languages are accepted, for instance, by bounded synchronized automata [20], and recursively enumerable languages are accepted by Turing machines [4]. And if it is well-known that regular languages or context-sensitive languages form a Boolean algebra, it is also well-known that it is not the case of context-free languages. Nevertheless, it was shown that various subclasses of context-free languages form Boolean algebras [17, 18, 5], as for example visibly pushdown languages with respect to a given pushdown alphabet [1]. Besides, pushdown automata model sequential computations. For parallel computations, a relevant family of automata consists of vector addition systems. Hence, the question arises of which Boolean algebras can be obtained from this family of automata.

[1] A word-automatic graph is a graph of which the vertex set is a regular language and each relation is recognized by a finite letter by letter synchronized transducer.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 48; pp. 48:1–48:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we consider trace automata. Their vertices are Mazurkiewicz traces [10] and they accept finite words. Traces bear the advantage of describing executions of concurrent systems [23]. We define the length of a trace as the length of its Foata normal form and we show that we can obtain Boolean algebras from any trace automata family $\mathcal{F}$ closed under level-length synchronization and level-length superposition (Theorem 34). These operations ensure the stability under intersection and difference of the class of recognized word languages. More precisely, we show that for any deterministic automaton $H \in \mathcal{F}$, the class of languages accepted by the deterministic automata in $\mathcal{F}$ length-reducible to H form a Boolean algebra of word languages.

Then, we apply the previous result to the family TrSuffix of trace suffix automata (with level-regular contexts), introduced in [16]. A trace suffix automaton is described by a finite set of rules of the form $\mathcal{W}(u \xrightarrow{a} v)$, where u and v are traces, a is a label and $\mathcal{W}$ is a level-regular trace language (*i.e.*, a language of traces of which the Foata normal forms form a regular word language). We show that this family satisfies the closure conditions stated above.

Lastly, we deduce that the subfamily $\text{TrSuffix}^{\text{VAS}}$ of trace suffix automata over trace monoids of which the dependence alphabet is the equality also satisfies the closure conditions stated above. Since $\text{TrSuffix}^{\text{VAS}}$ essentially corresponds to some vector addition systems (VAS), we obtain various Boolean algebras of word languages accepted by deterministic vector addition systems.

Related works. In [6, 7], Caucal and Rispal adapt Eilenberg's recognizability for languages [11] to infinite automata in order to obtain Boolean algebras. More precisely, in [7], they show how to obtain various Boolean algebras from any family of word automata (*i.e.*, automata of which vertices are words) closed under the operations of length synchronization and superposition. However, vector additions systems, seen as word automata (each trace is encoded by its Foata normal form), are not closed under length superposition.

Besides, in the literature, different types of VAS languages were considered [9, 12, 19, 22]: the labeling function may be free or not, λ-transitions (transitions labeled by the empty word) may be allowed or not, the set of final markings may be finite or equal to all accessible markings. In general, the various investigations focus on closure properties [12] and on the relationship with the other classical formal languages enumerated above [22, 21, 14]. Indeed, it is well known that VAS languages contain regular languages, are incomparable with context-free languages, but are context-sensitive [19]. The regularity (respectively the context-freeness) of some VAS languages is decidable [22] (respectively [21, 14]). But, if some type of VAS languages are closed under union and intersection, the complementation remains, to our knowledge, a challenging problem. In the usual terminology of VAS (or Petri nets), the VAS for which we prove that the languages form a Boolean algebra are labeled (*i.e.*, the labeling function is total and may not be injective), deterministic (from any marking, distinct transitions labeled by a same letter are not allowed) and equipped of a level-regular set of final vertices. Moreover, an action can be performed only with respect to a context that is assumed level-regular also.

2 Preliminaries

In this section, we give some preliminaries about automata and Mazurkiewicz traces. We use standard notations. In particular, the union of two disjoint sets A and B is denoted $A \mathbin{\dot\cup} B$.

2.1 Automata: definition and generalities

An automaton is given by a set of edges labeled by letters, plus initial and final vertices.

Let V be a set (of vertices), T be a set of symbols called *terminals* and $C = \{\iota, o\}$ be a set of *colors*. A T*-automaton* G over V is a subset of $V \times T \times V \ \cup \ C \times V$ of vertex set

$$V_G := \{v \in V \mid \exists\, a, u\ (u, a, v) \in G \ \vee \ (v, a, u) \in G\} \ \cup \ \{v \in V \mid \exists\, c\ (c, v) \in G\}$$

such that the set $T_G = \{a \in T \mid \exists\, u, v \in V, (u, a, v) \in G\}$ is finite. The automaton G is *finite* if its vertex set V_G is finite. Denote by $I_G := \{v \in V_G \mid (\iota, v) \in G\}$ the set of *initial vertices* of G and by $F_G := \{v \in V_G \mid (o, v) \in G\}$ the set of *final vertices* of G.

An element $(u, a, v) \in G$ is an *edge*. Its *label* is a, its *source* is u and its *target* is v. The notation $u \xrightarrow[G]{a} v$ (or $u \xrightarrow{a} v$ when G is understood) means $(u, a, v) \in G$. Any couple $(c, v) \in G$ is a vertex v *colored* by $c \in C$.

The automaton G is *deterministic* if I_G is reduced to a singleton and for each $a \in T$, if $(u \xrightarrow[G]{a} v$ and $u \xrightarrow[G]{a} v')$ then $v = v'$. Given a family $\mathcal{F}$ of automata, we denote by $\mathcal{F}_{\mathrm{det}}$ the subfamily of $\mathcal{F}$ consisting in deterministic automata.

A *path* in G of *source* u and *goal* v, labeled by a word $a_1 \ldots a_k \in T^*$, is a finite sequence of the form $u \xrightarrow[G]{a_1} u_1, \ldots, u_{k-1} \xrightarrow[G]{a_k} v$, with $u = v$ for $k = 0$. We denote by $u \xrightarrow[G]{a_1 \ldots a_k} v$ the existence of such a path. A path is *accepting* if its source is initial and its goal is final. The *language accepted* by an automaton G is the set $L(G)$ of words that label its accepting paths. The regular languages of T-words are the languages accepted by finite automata.

A *morphism* f from a T-automaton G into a T-automaton H is a mapping $f : V_G \to V_H$ such that

$$u \xrightarrow[G]{a} v \Longrightarrow f(u) \xrightarrow[H]{a} f(v) \quad \text{and} \quad (c, u) \in G \Longrightarrow (c, f(u)) \in H$$

If such a morphism exists, we write $G \xrightarrow{f} H$ (or just $G \to H$) and we say that G is f*-reducible* (or just *reducible*) to H. Moreover, if f is a bijection and $G \xrightarrow{f} H$ and $H \xrightarrow{f^{-1}} G$, then G and H are said to be f*-isomorphic*.

▶ **Lemma 1.** *Let G and H be automata. If $G \to H$, then $L(G) \subseteq L(H)$.*

Suppose there exists a length-mapping $|\ | : V \longrightarrow \mathbb{N}$. A morphism f is *length-preserving* if $|f(v)| = |v|$ for any $v \in V_G$. If such a morphism exists, we say that G is *length-reducible* to H and we write $G \xrightarrow{f}_{\ell} H$ (or just $G \to_{\ell} H$). The automata G and H are *length-isomorphic* if there exists a length-preserving morphism f such that G and H are f-isomorphic.

▶ **Example 2.** Let us consider the infinite binary tree $T_2 := \{Ku \xrightarrow{a} Kua \mid u \in \{a, b\}^*\} \ \cup \ \{Ku \xrightarrow{b} Kub \mid u \in \{a, b\}^*\} \ \cup \ \{(\iota, K)\} \ \cup \ \{o\} \times K\{a, b\}^*$ and the visibly pushdown automaton $\mathrm{Vis}(\{c\}, \varnothing, \{a, b\}) := \{\bot\top^n \xrightarrow{a,b} \bot\top^{n+1} \mid n \geqslant 0\} \ \cup \ \{\bot\top^n \xrightarrow{c} \bot\top^{n-1} \mid n \geqslant 1\} \ \cup \ \{\bot \xrightarrow{c} \bot\} \ \cup \ \{(\iota, \bot)\} \ \cup \ \{o\} \times \{\bot\top^n \mid n \geqslant 0\}$. The automaton T_2 is length-reducible to $\mathrm{Vis}(\{c\}, \varnothing, \{a, b\})$. See Figure 1.

2.2 Mazurkiewicz traces

Given a (finite) alphabet Σ, recall that Σ^* is the free monoid of finite Σ-words.

A *dependence relation* D on Σ is a reflexive and symmetric binary relation. The pair (Σ, D) is called a *dependence alphabet*. The complement of D is the *independence relation* $I := \Sigma^2 \backslash D$. The (Σ, D)-trace equivalence $\equiv_D$ is the least congruence on Σ^* such that

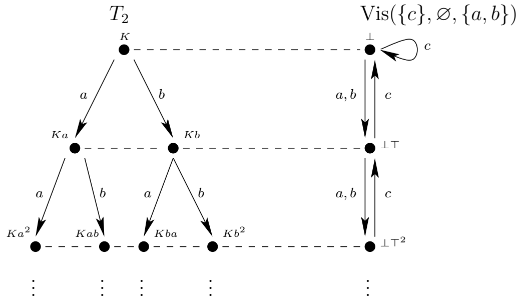


Figure 1 T_2 is length-reducible to $\mathrm{Vis}(\{c\}, \varnothing, \{a, b\})$ (Example 2).

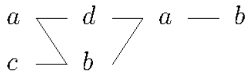


Figure 2 The Foata normal form of $[acbdab]$ (Example 4).

$(a, b) \in I \Rightarrow ab \equiv_D ba$. The (Σ, D)-trace of a word $w \in \Sigma^*$ is its $\equiv_D$-equivalence class. It is denoted $[w]$. Note that $\equiv_D$-equivalent Σ-words have the same length. The quotient monoid $\Sigma^* / \equiv_D$ is called the *trace monoid* of the dependence alphabet (Σ, D) and is denoted by $M(\Sigma, D)$. Its elements are called the *traces* over (Σ, D). The length of a trace t is the length of any word belonging to it and is denoted $|t|$. Denote by $\mathrm{Total}_\Sigma := \Sigma \times \Sigma$ the total dependence relation over Σ and by $\mathrm{Id}_\Sigma := \{(a, a) \mid a \in \Sigma\}$ the equality. Note that in case of $D = \mathrm{Total}_\Sigma$, the trace monoid $M(\Sigma, D)$ coincides with the free monoid Σ^*.

Consider the finite alphabet $I_D := \{A \subseteq \Sigma \mid \forall a_1 \neq a_2 \in A \ (a_1, a_2) \in I\}$ of independent subsets of Σ, and denote by $\Pi_{I_D} : I_D^* \to M(\Sigma, D)$ the canonical morphism defined by $\Pi_{I_D}(\varnothing) = [\varepsilon]$ and $\Pi_{I_D}(\{a_1, \cdots, a_n\}) = [a_1 \ldots a_n]$ $(n \geqslant 1)$. Consider the binary relation $\rhd$ on $I_D^- := I_D \setminus \{\varnothing\}$ defined by: $A \rhd B \iff \forall b \in B \ \exists a \in A \ (a, b) \in D$. The set of ${I_D^-}^*$-words of the form $A_1 \ldots A_p$ $(p \geqslant 0)$ such that $A_1 \rhd A_2 \rhd \cdots \rhd A_p$ is denoted $\mathbf{F}$. The surjective morphism Π_{I_D} is not injective. Indeed, suppose $\Sigma = \{a, b\}$ and aIb, then $\Pi_{I_D}(\{a, b\}) = \Pi_{I_D}(\{a\}\{b\})$. The following proposition expresses that each trace is encodable by a unique I_D^--word in $\mathbf{F}$.

▶ **Proposition 3** (Foata normal form, [10])**.** *Let $t \in M(\Sigma, D)$. There exists a unique I_D^--word $\lceil t \rceil_{\mathbf{F}} = A_1 \cdots A_p \in \mathbf{F}$ $(p \geqslant 0)$, the Foata normal form of t, such that $\Pi_{I_D}(A_1 \cdots A_p) = t$.*

▶ **Example 4.** Suppose $\Sigma = \{a, b, c, d\}$ and aIc, bId, cId. The Foata normal form of $t = [acbdab]$ (see Figure 2) is $\lceil t \rceil_{\mathbf{F}} = \{a, c\}\{b, d\}\{a\}\{b\}$.

The following lemma is straightforward.

▶ **Lemma 5** (Level automata)**.** *The set $\mathbf{F}$ of Foata normal forms is a regular word language over I_D^-.*

In general, $\lceil st \rceil_{\mathbf{F}}$ and $\lceil s \rceil_{\mathbf{F}} \lceil t \rceil_{\mathbf{F}}$ may be different. Indeed, suppose $D = \{(a, a), (b, b)\}$. If $s = [a]$ and $t = [ab]$, then $\lceil s \rceil_{\mathbf{F}} = \{a\}$, $\lceil t \rceil_{\mathbf{F}} = \{a, b\}$ and $\lceil st \rceil_{\mathbf{F}} = \{a, b\}\{a\}$. The following lemma expresses some compatibility between concatenation and Foata normal form.

▶ **Lemma 6.** *Let $s, t \in M(\Sigma, D)$ such that $\lceil s \rceil_{\mathbf{F}} = A_1 \cdots A_p$ $(p \geqslant 0)$. Then there exist $B_{i_1}, \ldots B_{i_k} \in I_D^-$ $(0 \leqslant k \leqslant p$ and $1 \leqslant i_1 < \cdots < i_k \leqslant p)$ and $C_1, \ldots, C_m \in I_D^-$ $(m \geqslant 0)$ such that $\lceil st \rceil_{\mathbf{F}} = A_1 \ldots (A_{i_1} \dot\cup B_{i_1}) \ldots (A_{i_k} \dot\cup B_{i_k}) \ldots A_p C_1 \ldots C_m$ and $\Pi_{I_D^-}(B_{i_1} \ldots B_{i_k} C_1 \ldots C_m) = t$.*

Proof. By induction on the length of t. ◀

In the following, given a trace t, we denote by $\|t\|$ the length of its Foata normal form.

A *trace automaton* is an automaton of which the vertices belong to some trace monoid $M(\Sigma, D)$ and accepts finite words (over an alphabet that may have no relation with Σ).

3 From word (suffix) automata to trace (suffix) automata

In this section, we recall how to obtain various Boolean algebras of deterministic context-free languages from word suffix automata [7]. Then we consider the notion of level-regular trace languages. This allows to define the family TrSuffix of trace suffix automata with level-regular contexts, an extension to traces of word suffix automata. Lastly, we will define (in terms of trace suffix automata) the vector addition systems from which we will obtain Boolean algebras (Section 5).

3.1 Boolean algebras from word suffix automata

Given a word language L over T, a non-empty family of languages $\mathcal{F}_L$ is a *Boolean algebra relative to* L if $L_1 \subseteq L$, $L - L_1 \in \mathcal{F}_L$ and $L_1 \cap L_2 \in \mathcal{F}_L$ for any $L_1, L_2 \in \mathcal{F}_L$. Observe that if $\mathcal{F}_L$ is a Boolean algebra relative to L then $\varnothing, L \in \mathcal{F}_L$.

A *word suffix automaton* is a finite union of automata of the form

$$W(u \xrightarrow{a} v) \;\cup\; \{\iota\} \times I \;\cup\; \{o\} \times F$$

where W, I and F are regular word languages (over a finite alphabet), u and v are words, $a \in T$ and $W(u \xrightarrow{a} v) = \{wu \xrightarrow{a} wv \mid w \in W\}$. We denote by Stack the family of word suffix automata. The languages accepted by Stack are the context-free languages [3].

In [7], Caucal and Rispal show how to obtain various Boolean algebras of deterministic context-free languages.

▶ **Theorem 7** ([7]). *Let H be a deterministic word suffix automaton of which the empty word is not a vertex. Then the class of languages $\mathrm{Rec}^{\ell}_{\mathrm{Stack}_{\mathrm{det}}}(H) = \{L(G) \mid G \in \mathrm{Stack}_{\mathrm{det}}, G \to_{\ell} H\}$ is a Boolean algebra relative to $L(H)$.*

In the following, we will consider the family TrSuffix of trace suffix automata with level-regular contexts, introduced in [16] (see Subsection 3.3). This family is an extension to Mazurkiewicz traces of word suffix automata (a word suffix automaton is just a trace suffix automaton with level-regular contexts over a trace monoid for which the dependence relation is total). We will also consider the subfamily $\mathrm{TrSuffix}^{\mathrm{VAS}}$ of vector addition systems and we will show how to obtain Boolean algebras from these, in the same vein as Theorem 7.

3.2 Level-regularity

In order to build the family TrSuffix of trace suffix automata with level-regular contexts, let us give some reminders about the notion of level-regular trace languages [16].

A (Σ, D)-*trace language* is a subset of $M(\Sigma, D)$. If $\mathcal{L}$ is a trace language, then $\bigcup \mathcal{L} = \{w \in \Sigma^* \mid [w] \in \mathcal{L}\}$. If L is a word language, then $[L]$ is the trace language defined by $[L] := \{[w] \in M(\Sigma, D) \mid w \in L\}$. In particular, $[\bigcup \mathcal{L}] = \mathcal{L}$ and $\bigcup [L] \supseteq L$.

Given a trace language $\mathcal{L} \subseteq M(\Sigma, D)$ and a trace $t \in M(\Sigma, D)$, the *right residual* (respectively the *left residual*) of $\mathcal{L}$ by t, $\mathcal{L}t^{-1}$ (respectively $t^{-1}\mathcal{L}$), is $\mathcal{L}t^{-1} := \{s \in M(\Sigma, D) \mid st \in \mathcal{L}\}$ (respectively $t^{-1}\mathcal{L} := \{s \in M(\Sigma, D) \mid ts \in \mathcal{L}\}$). The product of $\mathcal{L}$ by t (respectively the product of t by $\mathcal{L}$), $\mathcal{L}t$ (respectively $t\mathcal{L}$), is $\mathcal{L}t := \{st \in M(\Sigma, D) \mid s \in \mathcal{L}\}$ (respectively $t\mathcal{L} := \{ts \in M(\Sigma, D) \mid s \in \mathcal{L}\}$). If $\mathcal{L}_1, \mathcal{L}_2 \subseteq M(\Sigma, D)$ are trace languages, their product is the trace language $\mathcal{L}_1\mathcal{L}_2 := \{t_1t_2 \mid t_1 \in \mathcal{L}_1, t_2 \in \mathcal{L}_2\}$. In particular, $\mathcal{L}\varnothing = \varnothing\mathcal{L} = \varnothing$.

A trace language $\mathcal{L} \subseteq M(\Sigma, D)$ is *recognizable* if there exists a finite monoid N and a monoid morphism $\phi : M(\Sigma, D) \to N$ such that $\mathcal{L} = \phi^{-1}(\phi(\mathcal{L}))$. The class of recognizable trace languages is denoted by $\mathrm{Rec}(M(\Sigma, D))$.

▶ **Proposition 8** ([10]). *The following are equivalent:*

- *$\mathcal{L}$ is recognizable,*
- *$\bigcup \mathcal{L}$ is a regular word language,*
- *the set $\{t^{-1}\mathcal{L} \mid t \in M(\Sigma, D)\}$ of left residuals of $\mathcal{L}$ is finite,*
- *the set $\{\mathcal{L}t^{-1} \mid t \in M(\Sigma, D)\}$ of right residuals of $\mathcal{L}$ is finite.*

▶ Remark 9. In case of $D = \mathrm{Total}_\Sigma$, $\mathrm{Rec}(M(\Sigma, D)) = \mathrm{Reg}(\Sigma^*)$.

▶ **Proposition 10** ([10]). $\mathrm{Rec}(M(\Sigma, D))$ *is a Boolean algebra closed under concatenation.*

The trace language $[(ab)^*]$ with aIb is not recognizable since its union, the set of words over $\{a, b\}$ with the same number of occurences of a and b, is not regular. Nevertheless, the set $\{a, b\}^*$ of Foata normal forms of its elements is regular. This suggests considering a weaker notion of recognizability.

▶ **Definition 11.** *$\mathcal{L} \subseteq M(\Sigma, D)$ is level-regular if the word language $\lceil \mathcal{L} \rceil_{\mathbf{F}}$ is regular.*

Since $\lceil \mathcal{L} \rceil_{\mathbf{F}} = \Pi_{I_D}^{-1}(\mathcal{L}) \cap \mathbf{F}$, any recognizable trace language is level-regular. Note also that any level-regular trace language is a rational subset of the monoid $M(\Sigma, D)$. Denote by $\mathrm{LevelReg}(M(\Sigma, D))$ the class of level-regular languages of the trace monoid $M(\Sigma, D)$.

▶ **Proposition 12** ([16]). $\mathrm{LevelReg}(M(\Sigma, D))$ *is a Boolean algebra.*

The class $\mathrm{LevelReg}(M(\Sigma, D))$ is not closed under concatenation [16]. Nevertheless, multiplication and residuation of trace languages on the right by a trace preserves the level-regularity.

▶ **Lemma 13.** *Let $\mathcal{L} \in \mathrm{LevelReg}(M(\Sigma, D))$ and $t \in M(\Sigma, D)$. The trace languages $\mathcal{L}t^{-1}$ and $\mathcal{L}t$ are level-regular.*

Given a word language L, we denote by $\mathrm{Pref}(L) := \{u \mid \exists\ v\ uv \in L\}$ the set of prefixes of words in L. From the lemma below, it follows, in particular, that the set of prefixes of Foata normal forms of a level-regular trace language is a regular word language.

▶ **Lemma 14.** *Let $\mathcal{L} \in \mathrm{LevelReg}(M(\Sigma, D))$. Then $\Pi_{I_D}(\mathrm{Pref}\,\lceil \mathcal{L} \rceil_{\mathbf{F}}) \in \mathrm{LevelReg}(M(\Sigma, D))$.*

3.3 Trace suffix automata with level-regular contexts

We consider trace suffix automata with level-regular contexts [16]. These generalize both word suffix automata (see Subsection 3.1) and vector addition systems (see the following subsection).

A *trace suffix automaton (with level-regular contexts)* over a trace monoid $M(\Sigma, D)$ is of the form

$$G = \bigcup_{1 \leqslant i \leqslant n} \mathcal{W}_i(u_i \xrightarrow{a_i} v_i) \ \cup\ \{\iota\} \times \mathcal{I}_G \ \cup\ \{o\} \times \mathcal{F}_G$$

where $\mathcal{W}_i, \mathcal{I}_G, \mathcal{F}_G \in \mathrm{LevelReg}(M(\Sigma, D)), u_i, v_i \in M(\Sigma, D)$, $a_i \in T$ and $\mathcal{W}_i(u_i \xrightarrow{a_i} v_i) = \{w_i u_i \xrightarrow{a_i} w_i v_i \mid w_i \in \mathcal{W}_i\}$ $(1 \leqslant i \leqslant n)$. The trace language $\mathcal{W}_i$ is called the *context* of the *rewriting rule* $\mathcal{W}_i(u_i \xrightarrow{a_i} v_i)$ $(1 \leqslant i \leqslant n)$. Denote by TrSuffix the family of trace suffix automata.

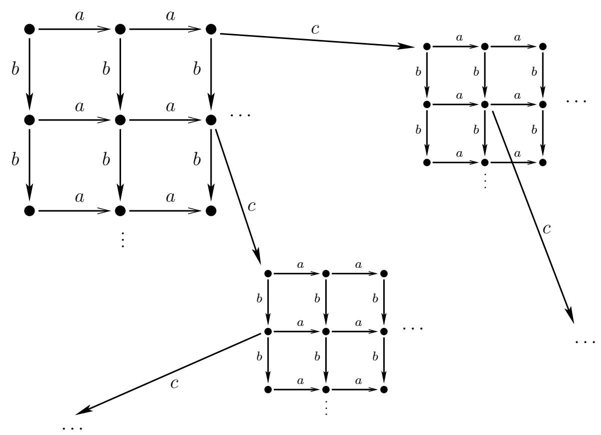


Figure 3 The infinite quarter grid tree is a trace suffix automaton (see Example 15).

▶ **Example 15** (Infinite quarter grid tree)**.** See Figure 3. Consider the following trace suffix automaton with level-regular contexts, where the independence relation is $\{(a,b),(b,a)\}$.

$[\bot\{a,b,c\}^*]([\varepsilon] \xrightarrow{a} [a]) \ \cup\ [\bot\{a,b,c\}^*]([\varepsilon] \xrightarrow{b} [b]) \cup\ [\bot\{a,b,c\}^*]([\varepsilon] \xrightarrow{c} [c])$

In particular, it was shown in [16] that its first-order theory with reachability is decidable though it is not a ground term rewriting graph [15].

The trace suffix automata for total dependence relations are the word suffix automata. Thus regular languages and context-free languages are accepted by trace suffix automata. On the other hand:

▶ **Proposition 16** ([16, 20])**.** *The languages accepted by the trace suffix automata are context-sensitive.*

The previous proposition relies on the fact that trace suffix automata are actually word-automatic automata [16] and the latter accept the context-sensitive languages [20].

3.4 Vector addition systems

Here, a *vector addition system* G over Σ is a trace suffix automaton over the trace monoid $M(\Sigma, \mathrm{Id}_\Sigma)$. Denote by $\mathrm{TrSuffix}^{\mathrm{VAS}}$ the family of vector addition systems. The family $\mathrm{TrSuffix}^{\mathrm{VAS}}$ is a subfamily of TrSuffix.

▶ **Example 17.** Observe that a vector addition system over a singleton alphabet is a word suffix automaton since the equality relation over such an alphabet coincides with the total relation. Let (T_{-1}, T_0, T_1) a triple of disjoint finite alphabets. The trace suffix automaton $\mathrm{Vis}^{\mathrm{VAS}\,\vee\,\mathrm{Stack}}(T_{-1}, T_0, T_1)$ over $M(\{\top\}, \{(\top,\top)\})$ defined below is a vector addition system over the singleton alphabet $\{\top\}$. It is length-isomorphic to a word suffix automaton. The Boolean algebra $\mathrm{Rec}^{\ell}_{\mathrm{Stack}_{\mathrm{det}}}(\mathrm{Vis}^{\mathrm{VAS}\,\vee\,\mathrm{Stack}}(T_{-1}, T_0, T_1))$ (see Theorem 7) is the family of visibly pushdown languages with respect to (T_{-1}, T_0, T_1) ([1]).

$$\begin{aligned}\mathrm{Vis}^{\mathrm{VAS}\,\vee\,\mathrm{Stack}}(T_{-1}, T_0, T_1) := &\bigcup_{\lambda\in T_1} [\top^+]([\varepsilon] \xrightarrow{\lambda} [\top]) \ \dot\cup \ \bigcup_{\lambda\in T_0} [\top^+]([\varepsilon] \xrightarrow{\lambda} [\varepsilon]) \\ &\dot\cup \ \bigcup_{\lambda\in T_{-1}} [\top^+]([\top] \xrightarrow{\lambda} [\varepsilon]) \ \dot\cup \ \bigcup_{\lambda\in T_{-1}} [\top]([\varepsilon] \xrightarrow{\lambda} [\varepsilon]) \ \dot\cup \ \{\iota\} \times \{[\top]\} \ \dot\cup \ \{o\} \times [\top^+]\end{aligned}$$

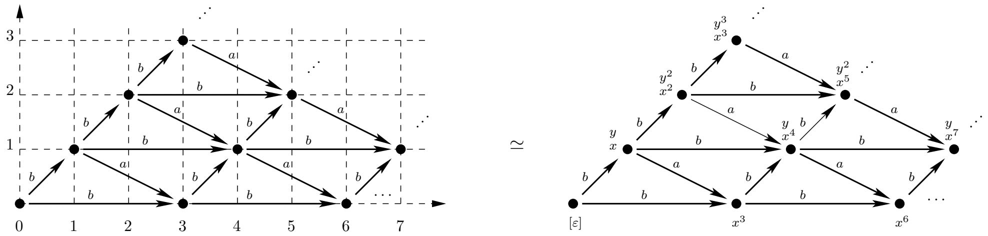


Figure 4 A vector addition system from a finite set of integer vectors.

In the previous example, the length variation of two adjacent vertices is almost 1. It is no more the case for the following example that shows, in particular, how to obtain a vector addition system from a finite set of integer vectors of the same dimension.

▶ **Example 18.** Consider the finite set $A := \{\binom{2}{-1}, \binom{1}{1}, \binom{3}{0}\}$ of 2-dimensional integer vectors and the automaton G_A over $\mathbb{N}^2$, defined by

$$G_A := \{v \xrightarrow{l(a)} v' \mid v, v' \in \mathbb{N}^2, a \in A, v + a = v'\}$$

where $l : A \longrightarrow T$ is the labeling function that maps $\binom{1}{1}$ and $\binom{3}{0}$ to the terminal b and $\binom{2}{-1}$ to the terminal a. The automaton G_A is isomorphic to the vector addition system over the alphabet $\{x, y\}$ defined by the following rules

$$M([y] \xrightarrow{a} [xx]) \ \cup \ M([\varepsilon] \xrightarrow{b} [xy]) \ \cup \ M([\varepsilon] \xrightarrow{b} [xxx])$$

where M denotes the trace monoid $M(\{x, y\}, \mathrm{Id}_{\{x,y\}})$. See Figure 4.

4 Level-length synchronization and Boolean algebras from trace automata

In this section, we show how to obtain various Boolean algebras of word languages from the family TrSuffix (Theorem 37). Actually, this will be deduced from a more general result, since we give sufficient conditions for any family of trace automata to define Boolean algebras (Theorem 34). After considering a notion of synchronization of two traces belonging to disjoint trace monoids, we define their level-length synchronization. Then we define the level-length synchronization and superposition of two trace automata (Definition 23 and 27) and we show that under some relevant assumptions, they accept respectively the intersection and the difference (Lemmas 25 and 33). As a consequence, we show how to obtain a Boolean algebra of word languages from any trace automata family closed under level-length synchronization and superposition and from a deterministic automaton in this family (Theorem 34). Finally, we show that the family TrSuffix is closed under level-length synchronization and superposition.

Let $M(\Sigma_1, D_1)$ and $M(\Sigma_2, D_2)$ be trace monoids such that $\Sigma_1 \cap \Sigma_2 = \varnothing$.

Let $s \in M(\Sigma_1, D_1)$ and $t \in M(\Sigma_2, D_2)$.

▶ **Definition 19.** *The synchronization $s \parallel t$ of s and t is the product st in the trace monoid $M(\Sigma_1 \mathbin{\dot\cup} \Sigma_2, D_1 \ \cup \ D_2)$.*

▶ **Definition 20.** *The level-length synchronization $s \parallel_= t$ of s and t is $s \parallel t$ if $\|s\| = \|t\|$ and is not defined otherwise.*

We also define $s \parallel_{\geqslant} t := s \parallel t$ if $\|s\| \geqslant \|t\|$ and $s \parallel_{\leqslant} t := s \parallel t$ if $\|s\| \leqslant \|t\|$.

Given $\mathcal{L}_1 \subseteq M(\Sigma_1, D_1)$ and $\mathcal{L}_2 \subseteq M(\Sigma_2, D_2)$, we define
$\mathcal{L}_1 \parallel \mathcal{L}_2 := \{t_1 \parallel t_2 \mid t_1 \in \mathcal{L}_1, t_2 \in \mathcal{L}_2\}$ and $\mathcal{L}_1 \parallel_{=} \mathcal{L}_2 := \{t_1 \parallel_{=} t_2 \mid t_1 \in \mathcal{L}_1, t_2 \in \mathcal{L}_2\}$,
$\mathcal{L}_1 \parallel_{\geqslant} \mathcal{L}_2 := \{t_1 \parallel_{\geqslant} t_2 \mid t_1 \in \mathcal{L}_1, t_2 \in \mathcal{L}_2\}$ and $\mathcal{L}_1 \parallel_{\leqslant} \mathcal{L}_2 := \{t_1 \parallel_{\leqslant} t_2 \mid t_1 \in \mathcal{L}_1, t_2 \in \mathcal{L}_2\}$.
These synchronized trace languages remain level-regular if $\mathcal{L}_1$ and $\mathcal{L}_2$ are.

▶ **Lemma 21.** *Let $\mathcal{L}_1 \in \mathrm{LevelReg}(M(\Sigma_1, D_1))$ and $\mathcal{L}_2 \in \mathrm{LevelReg}(M(\Sigma_2, D_2))$. The following trace languages are level-regular: $\mathcal{L}_1 \parallel \mathcal{L}_2$, $\mathcal{L}_1 \parallel_{\geqslant} \mathcal{L}_2$, $\mathcal{L}_1 \parallel_{\leqslant} \mathcal{L}_2$, $\mathcal{L}_1 \parallel_{=} \mathcal{L}_2$.*

If a synchronized trace language is level-regular, then so are its projections. Let us make explicit what such a projection is. For $i \in \{1, 2\}$, denoting $\bar{i} := 3 - i$, we consider the morphism π_i from $M(\Sigma_1 \dot\cup \Sigma_2, D_1 \dot\cup D_2)$ into $M(\Sigma_i, D_i)$ defined by $\pi_i([a]) = [a]$ if $a \in \Sigma_i$ and $\pi_i([a]) = [\varepsilon]$ if $a \in \Sigma_{\bar{i}}$. Given $t \in M(\Sigma_1 \dot\cup \Sigma_2, D_1 \dot\cup D_2)$, there exists a unique couple $(t_1, t_2) \in M(\Sigma_1, D_1) \times M(\Sigma_2, D_2)$ such that $t = t_1 t_2$. This couple is given by $t_1 = \pi_1(t)$ and $t_2 = \pi_2(t)$. The morphism π_i naturally extends to trace languages.

▶ **Lemma 22.** *Let $\mathcal{L} \in \mathrm{LevelReg}(M(\Sigma_1 \dot\cup \Sigma_2, D_1 \dot\cup D_2))$. The trace languages $\pi_1(\mathcal{L})$ and $\pi_2(\mathcal{L})$ are level-regular.*

Now, we consider the level-length synchronization of two automata. It is an automaton over the level-length synchronization of vertices of these automata.

▶ **Definition 23.** *Let G_1 and G_2 be trace automata over respectively $M(\Sigma_1, D_1)$ and $M(\Sigma_2, D_2)$. Their level-length synchronization $G_1 \parallel_{=} G_2$ is the trace automaton over $M(\Sigma_1 \dot\cup \Sigma_2, D_1 \dot\cup D_2)$ defined by*

$$\begin{aligned} G_1 \parallel_{=} G_2 := \{p_1 \parallel_{=} p_2 \xrightarrow{a} q_1 \parallel_{=} q_2 \mid p_1 \xrightarrow[G_1]{a} q_1, p_2 \xrightarrow[G_2]{a} q_2\} \\ \cup \ \{\iota\} \times (I_{G_1} \parallel_{=} I_{G_2}) \ \cup \ \{o\} \times (F_{G_1} \parallel_{=} F_{G_2}) \end{aligned}$$

▶ **Lemma 24.** $G_1 \parallel_{=} G_2 \xrightarrow{\pi_i}_{\ell} G_i$ $(i \in \{1, 2\})$.

If two automata are length-reducible to a same deterministic one, then the intersection of their languages is accepted by their level-length synchronization.

▶ **Lemma 25.** *If H is a deterministic trace automaton such that $G_1 \to_{\ell} H$ and $G_2 \to_{\ell} H$, then $L(G_1 \parallel_{=} G_2) = L(G_1) \cap L(G_2)$.*

Proof. Let $u \in L(G_1) \cap L(G_2)$ be a word. By Lemma 1 and since $G_1 \to_{\ell} H$, there exists an accepting path labeled by u in H. Since H is deterministic, this path is unique. Then, for any two paths in G_1 and G_2 accepting u, the sequences of lengths of vertices of these paths are same, because they are the same as the sequence of lengths of the path accepting u in H. Hence, u labels an accepting path in $G_1 \parallel_{=} G_2$. The other inclusion is straightforward. ◀

If two deterministic automata are length-reducible to a same deterministic one, then their level-length synchronization is also a deterministic automaton.

▶ **Lemma 26.** *If G_1, G_2 and H are deterministic trace automata such that $G_i \to_{\ell} H$ $(i \in \{1, 2\})$, then $G_1 \parallel_{=} G_2$ is a deterministic automaton.*

We are now introducing the level-length superposition $G \mathbin{/\!\!/^{\sharp}} H$ of trace automata. When restricted to deterministic automata and if $G \to_{\ell} H$, this accepts the difference language $L(H) - L(G)$ (Lemma 33). Note that to accept the difference, we may consider simpler operations. The level-length superposition bears the advantage of preserving the boundedness of the degree. Indeed, we will consider families of automata of bounded degree and in order to obtain Boolean algebras, we will require these families to be closed under level-length superposition.

Given a word language L and a word u, we write $u \sqsubseteq L$ if u is a prefix of a word in L. A trace automaton is $[\varepsilon]$-*free* if $[\varepsilon]$ is not a vertex.

▶ **Definition 27.** *Let G and H be $[\varepsilon]$-free trace automata over respectively $M(\Sigma_1, D_1)$ and $M(\Sigma_2, D_2)$ and $\sharp \notin \Sigma_1 \mathbin{\dot\cup} \Sigma_2$. The level-length superposition of G on H is the trace automaton $G \mathbin{/\!\!/^{\sharp}} H$ over $M(\Sigma_1 \mathbin{\dot\cup} \{\sharp\} \mathbin{\dot\cup} \Sigma_2, D_1 \mathbin{\dot\cup} \{(\sharp,\sharp)\} \mathbin{\dot\cup} D_2)$ defined by*

$$\begin{array}{rl}
 & G \mathbin{/\!\!/^{\sharp}} H := \\
(1) & \{p \parallel_= s \xrightarrow{a} q \parallel_= t \mid p \xrightarrow[G]{a} q, s \xrightarrow[H]{a} t\} \\
(2) & \cup \{\iota\} \times (I_G \parallel_= I_H) \\
(3) & \cup \{o\} \times ((V_G - F_G) \parallel_= F_H) \\
(4) & \cup \{p \parallel_= s \xrightarrow{a} p[\sharp] \parallel t \mid s \xrightarrow[H]{a} t, \|s\| \leqslant \|t\|, p \in V_G, \neg\exists p' (p \xrightarrow[G]{a} p' \ \wedge \|p'\| = \|t\|)\} \\
(5) & \cup \{p \parallel_= s \xrightarrow{a} q[\sharp] \parallel_= t \mid s \xrightarrow[H]{a} t, \|s\| > \|t\|, p \in V_G, \neg\exists p' (p \xrightarrow[G]{a} p' \ \wedge \|p'\| = \|t\|), \lceil q \rceil_{\mathbf{F}} \sqsubseteq \lceil p \rceil_{\mathbf{F}}\} \\
(6) & \cup \{p[\sharp] \parallel_{\leqslant} s \xrightarrow{a} p[\sharp] \parallel t \mid s \xrightarrow[H]{a} t, \|s\| \leqslant \|t\|, \lceil p \rceil_{\mathbf{F}} \sqsubseteq \lceil V_G \rceil_{\mathbf{F}}\} \\
(7) & \cup \{p[\sharp] \parallel_{\leqslant} s \xrightarrow{a} p[\sharp] \parallel_{\leqslant} t \mid s \xrightarrow[H]{a} t, \|s\| > \|t\|, \lceil p \rceil_{\mathbf{F}} \sqsubseteq \lceil V_G \rceil_{\mathbf{F}}\} \\
(8) & \cup \{p[\sharp] \parallel_{\leqslant} s \xrightarrow{a} q[\sharp] \parallel_= t \mid s \xrightarrow[H]{a} t, \|s\| > \|t\|, \|t\| < \|p[\sharp]\|, \lceil p \rceil_{\mathbf{F}} \sqsubseteq \lceil V_G \rceil_{\mathbf{F}}, \lceil q \rceil_{\mathbf{F}} \sqsubseteq \lceil p \rceil_{\mathbf{F}}\} \\
(9) & \cup \{o\} \times (\{p[\sharp] \parallel_{\leqslant} s \mid \lceil p \rceil_{\mathbf{F}} \sqsubseteq \lceil V_G \rceil_{\mathbf{F}}, s \in F_H\}) \\
(10) & \cup \{\iota\} \times (\{[\sharp] \parallel s \mid s \in I_H, \neg(\exists p \in I_G \ \|p\| = \|s\|)\})
\end{array}$$

Let us give some more explanations about this definition. First, observe that there are two kinds of vertices: those in which $\sharp$ occurs and the other ones. Then, observe that the edges between non $\sharp$-vertices are those of $G \parallel_= H$. Note that the edges at lines (4) and (5) are the only ones between $\sharp$-vertices and non $\sharp$-vertices, and that they leave the set $V_{G \parallel_= H}$ of non $\sharp$-vertices. Thus, once a path leaves $V_{G \parallel_= H}$, it never returns.

Given $t \in M(\Sigma_1 \mathbin{\dot\cup} \{\sharp\} \mathbin{\dot\cup} \Sigma_2, D_1 \mathbin{\dot\cup} D_2)$, we denote by $\pi_1(t)$ (respectively $\pi_2(t)$) the unique trace in $M(\Sigma_1 \mathbin{\dot\cup} \{\sharp\}, D_1)$ (respectively $M(\Sigma_2, D_2)$) such that $t = \pi_1(t)\pi_2(t)$.

For each edge $s \xrightarrow[H]{a} t$ and each vertex x of $G \mathbin{/\!\!/^{\sharp}} H$, H-projecting on s (*i.e.*, $\pi_2(x) = s$), there exists an edge $x \xrightarrow[G /\!\!/^{\sharp} H]{a} y$ with y H-projecting on t (*i.e.*, $\pi_2(y) = t$). Since on the other hand, any initial vertex of H lifts to an initial vertex of $G \mathbin{/\!\!/^{\sharp}} H$ (lines (2) and (10)), it follows that any initial path in H (an *initial path* is a path of which the source is an initial vertex) lifts to an initial path of $G \mathbin{/\!\!/^{\sharp}} H$.

▶ **Lemma 28.** *Any word that labels an initial path in H, also labels an initial path in $G \mathbin{/\!\!/^{\sharp}} H$.*

Lastly, note that we preserve the fact that the level-length of any vertex is given by the level-length of its H-component (see Lemma 30). In particular, the first component is not longer than the second one.

It will be relevant to discuss about the final vertices of $G \mathbin{/\!\!/^{\sharp}} H$ only under some more assumptions about G and H (Lemma 30, Corollary 32 and Lemma 33).

▶ **Example 29.** Let us consider the automata described in Example 2. The unique initial vertex of $T_2 \mathbin{/\!\!/^{\sharp}} \mathrm{Vis}(\{c\}, \varnothing, \{a,b\})$ is $K \parallel_= \bot$. The final vertices of $T_2 \mathbin{/\!\!/^{\sharp}} \mathrm{Vis}(\{c\}, \varnothing, \{a,b\})$ are these for which the number of occurrences of $\sharp$ is 1. See Figure 5 for the restriction of $T_2 \mathbin{/\!\!/^{\sharp}} \mathrm{Vis}(\{c\}, \varnothing, \{a,b\})$ to the vertices accessible from the initial vertex and co-accessible from final vertices.

▶ **Lemma 30.** *$G \mathbin{/\!\!/^{\sharp}} H \xrightarrow{\pi_2}_{\ell} H$.*

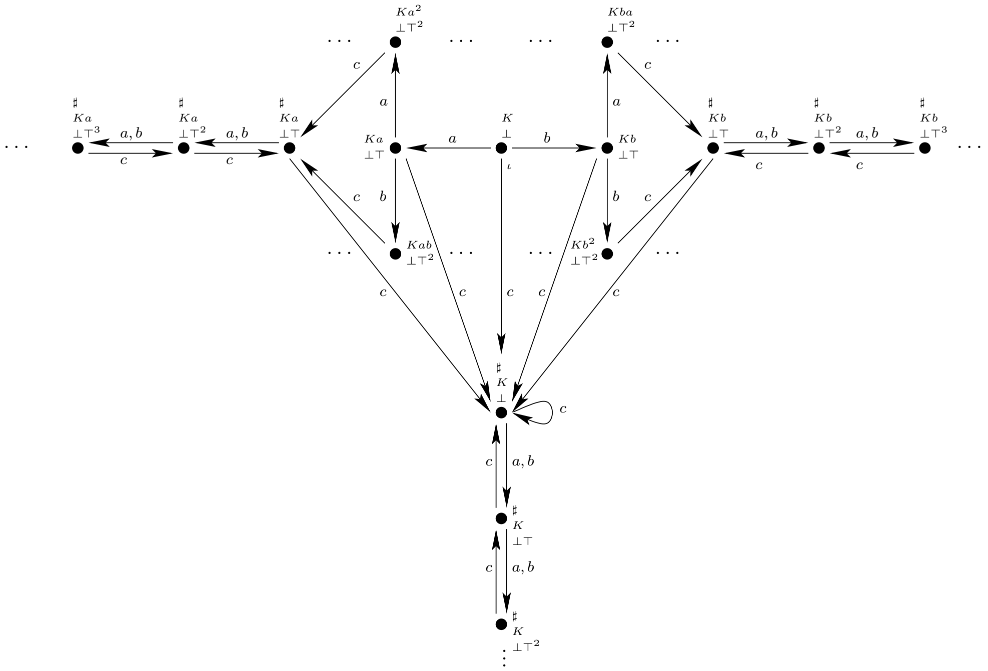


Figure 5 Trimmed level-length superposition of T_2 on $\mathrm{Vis}(\{c\}, \varnothing, \{a, b\})$ (see Example 29).

By Lemma 1, $L(G \mathbin{/\!\!/^\sharp} H) \subseteq L(H)$. That is, if a word labels an accepting path in $G \mathbin{/\!\!/^\sharp} H$ then it also labels an accepting path in H. The converse is not true in general, as we will see in Lemma 33.

The level-length superposition preserves determinism.

▶ **Lemma 31.** *If G and H are deterministic, then $G \mathbin{/\!\!/^\sharp} H$ is deterministic.*

Observe that the automaton described at lines (1), (2) and (3) (Definition 27) is the level-length synchronization of H and the automaton obtained from G by declaring final the non final vertices and *vice-versa*. By Lemma 25 and since obviously $H \to_\ell H$, we deduce the following corollary.

▶ **Corollary 32.** *If G and H are deterministic, $G \to_\ell H$ and w is a word that labels an initial path in $G \mathbin{\|_=} H$, then $w \in L(G) \cap L(H) \iff w \notin L(G \mathbin{/\!\!/^\sharp} H)$.*

The level-length superposition accepts the difference.

▶ **Lemma 33.** *If G and H are deterministic and $G \to_\ell H$, then $L(G \mathbin{/\!\!/^\sharp} H) = L(H) - L(G)$.*

Proof. By Lemmas 30, 31 and Corollary 32, it remains to show that any word $w \in L(H) - L(G)$ that does not label any initial path in $G \mathbin{\|_=} H$ is accepted by $G \mathbin{/\!\!/^\sharp} H$. The unique initial path in the deterministic automaton $G \mathbin{/\!\!/^\sharp} H$ labeled by w is accepting since its goal is of the form $p[\sharp] \mathbin{\|_\leqslant} s$ $(s \in F_H)$ and any such vertex is final in $G \mathbin{/\!\!/^\sharp} H$ (line (9) in Definition 27). ◀

Let us apply Lemmas 25 and 33. From any trace automata family closed under level-length synchronization and superposition, we obtain various Boolean algebras of word languages from the deterministic automata of this family.

▶ **Theorem 34.** *Let $\mathcal{F}$ be a family of trace automata closed under level-length synchronization and superposition and $H \in \mathcal{F}_{\mathrm{det}}$ $[\varepsilon]$-free. Then the class of languages $\mathrm{Rec}^{\ell}_{\mathcal{F}_{\mathrm{det}}}(H) = \{L(G) \mid G \in \mathcal{F}_{\mathrm{det}}, G \to_\ell H\}$ is a Boolean algebra relative to $L(H)$.*

Let us show that Theorem 34 applies to the family TrSuffix.

▶ **Proposition 35.** *The family* TrSuffix *is closed under level-length synchronization.*

Proof (Sketch). First, observe that given $G_1, G_2 \in \mathrm{TrSuffix}$, the sets of initial vertices and final vertices of $G_1 \parallel_= G_2$ are level-regular by Lemma 21. Indeed, the level-length synchronization of level-regular languages remains level-regular. Then, since the operation $\parallel_=$ is distributive over union, it suffices to suppose that G_1 (respectively G_2) is of the form $\mathcal{W}_1(u_1 \xrightarrow{a} v_1)$ over $M(\Sigma_1, D_1)$ (respectively $\mathcal{W}_2(u_2 \xrightarrow{b} v_2)$ over $M(\Sigma_2, D_2)$) with $\mathcal{W}_i \in \mathrm{LevelReg}(M(\Sigma_i, D_i)), u_i, v_i \in M(\Sigma_i, D_i)$ $(i \in \{1,2\})$. If $a \neq b$, then $G_1 \parallel_= G_2 = \varnothing$. Suppose $a = b$. We show that $G_1 \parallel_= G_2 := \{p_1 \parallel_= p_2 \xrightarrow{a} q_1 \parallel_= q_2 \mid p_1 \xrightarrow[G_1]{a} q_1, p_2 \xrightarrow[G_2]{a} q_2\}$ is equal to $\big((\mathcal{W}_1 u_1 \parallel_= \mathcal{W}_2 u_2)(u_1 \parallel u_2)^{-1} \cap (\mathcal{W}_1 v_1 \parallel_= \mathcal{W}_2 v_2)(v_1 \parallel v_2)^{-1}\big)\,(u_1 \parallel u_2 \xrightarrow{a} v_1 \parallel v_2)$. Since $(\mathcal{W}_1 u_1 \parallel_= \mathcal{W}_2 u_2)(u_1 \parallel u_2)^{-1} \cap (\mathcal{W}_1 v_1 \parallel_= \mathcal{W}_2 v_2)(v_1 \parallel v_2)^{-1}$ is level-regular by Lemmas 13, 21 and Proposition 12, this proves the proposition. ◀

▶ **Proposition 36.** *The family* TrSuffix *is closed under level-length superposition.*

Proof (Sketch). Since $G \mathbin{/\!\!/^{\sharp}} (H_1 \cup H_2) = G \mathbin{/\!\!/^{\sharp}} H_1 \;\cup\; G \mathbin{/\!\!/^{\sharp}} H_2$, it suffices to consider $G = \bigcup_{1 \leqslant i \leqslant n} \mathcal{W}_i(u_i \xrightarrow{a_i} v_i) \;\cup\; \{\iota\} \times \mathcal{I}_G \;\cup\; \{o\} \times \mathcal{F}_G$ over $M(\Sigma_1, D_1)$ and $H = \mathcal{Z}(x \xrightarrow{a} y) \;\cup\; \{\iota\} \times \mathcal{I}_H \;\cup\; \{o\} \times \mathcal{F}_H$ over $M(\Sigma_2, D_2)$. We may assume that the level-length variation of the rules $\mathcal{W}_i(u_i \xrightarrow{a_i} v_i)$ $(1 \leqslant i \leqslant n)$ and $\mathcal{Z}(x \xrightarrow{a} y)$ is constant and is equal respectively to δ_i and δ_H. We show that the sets of initial and final vertices of $G \mathbin{/\!\!/^{\sharp}} H$ correspond to level-regular trace languages. Then, by distinguishing the cases $\delta_H \geqslant 0$ and $\delta_H < 0$, we show that the sets of edges constituting $G \mathbin{/\!\!/^{\sharp}} H$ (Definition 27) can be described as trace suffix automata with level-regular contexts. In each case, the level-regularity of the trace languages considered (for initial and final vertices and for the contexts of the rewriting rules) will be ensured by Proposition 12 and Lemmas 13, 14, 21, 22. ◀

Let us apply Theorem 34 and Propositions 35 and 36.

▶ **Theorem 37.** *Let H be an $[\varepsilon]$-free deterministic trace suffix automaton. Then* $\mathrm{Rec}^{\ell}_{\mathrm{TrSuffix}_{\mathrm{det}}}(H) = \{L(G) \mid G \in \mathrm{TrSuffix}_{\mathrm{det}}, G \to_{\ell} H\}$ *is a Boolean algebra relative to* $L(H)$.

5 Vector addition systems

By Theorem 34 and since we will prove that $\mathrm{TrSuffix}^{\mathrm{VAS}}$ is closed under level-length synchronization and superposition, we obtain various Boolean algebras of word languages accepted by deterministic vector addition systems.

▶ **Theorem 38.** *Let* $H \in \mathrm{TrSuffix}^{\mathrm{VAS}}_{\mathrm{det}}$ *$[\varepsilon]$-free. Then* $\mathrm{Rec}^{\ell}_{\mathrm{TrSuffix}^{\mathrm{VAS}}_{\mathrm{det}}}(H) = \{L(G) \mid G \in \mathrm{TrSuffix}^{\mathrm{VAS}}_{\mathrm{det}}, G \to_{\ell} H\}$ *is a Boolean algebra relative to* $L(H)$.

Proof. Given Theorem 34, it suffices to show that $\mathrm{TrSuffix}^{\mathrm{VAS}}$ is closed under level-length synchronization and superposition. Consider $G_1, G_2 \in \mathrm{TrSuffix}^{\mathrm{VAS}}$ with G_1 over Σ_1, G_2 over Σ_2, $\Sigma_1 \cap \Sigma_2 = \varnothing$ and $\sharp \notin \Sigma_1 \mathbin{\dot\cup} \Sigma_2$. Then observe that $G_1 \parallel_= G_2$ (respectively $G_1 \mathbin{/\!\!/^{\sharp}} G_2$) is a trace suffix automaton over $\Sigma_1 \mathbin{\dot\cup} \Sigma_2$ (respectively over $\Sigma_1 \mathbin{\dot\cup} \{\sharp\} \mathbin{\dot\cup} \Sigma_2$). ◀

▶ **Example 39.** The Boolean algebra $\mathrm{Rec}^{\ell}_{\mathrm{TrSuffix}^{\mathrm{VAS}}_{\mathrm{det}}}(\mathrm{Vis}^{\mathrm{VAS} \vee \mathrm{Stack}}(\varnothing, \varnothing, \{a\}))$ is the Boolean algebra of regular languages over the singleton alphabet $\{a\}$.

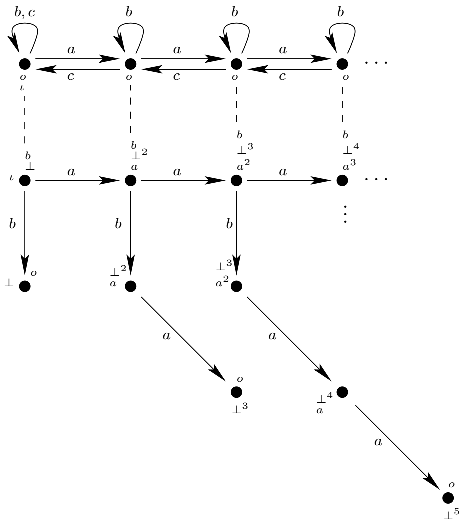


Figure 6 A trace suffix automaton that accepts $\{a^nba^n \mid n \geqslant 0\}$ (Example 40).

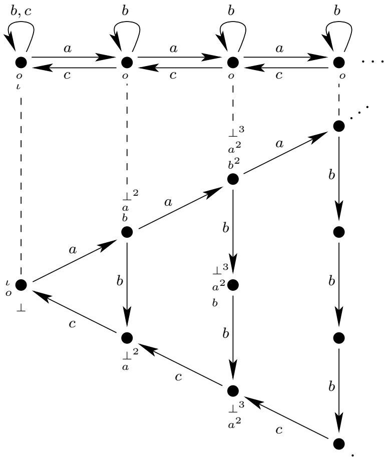


Figure 7 A trace suffix automaton that accepts $\{a^nb^nc^n \mid n \geqslant 0\}^*$ (Example 41).

▶ **Example 40.** It is well-known that the language $L = \{a^nba^n \mid n \geqslant 0\}$, while being context-free, is not a visibly pushdown language [1]. Consider the following vector addition system G over $\{\perp, a, b\}$ defined by

$$\begin{aligned} G := [(\perp a)^*\perp]([b] \xrightarrow{b} [\varepsilon]) \;\cup\; [b(\perp a)^*\perp]([\varepsilon] \xrightarrow{a} [\perp a]) \;\cup\; [(\perp a)^*\perp^+]([a] \xrightarrow{a} [\perp]) \\ \cup\; \{\iota\} \times \{[\perp b]\} \;\cup\; \{o\} \times [\perp^*] \end{aligned}$$

We have $L = L(G) \in \mathrm{Rec}^{\ell}_{\mathrm{TrSuffix}^{\mathrm{VAS}}_{\mathrm{det}}}(\mathrm{Vis}^{\mathrm{VAS} \vee \mathrm{Stack}}(\varnothing, \{b\}, \{a\}))$. See Figure 6.

▶ **Example 41.** It is well-known that the language $L = \{a^nb^nc^n \mid n \geqslant 0\}^*$ is not context-free. Consider the following vector addition system G over $\{\perp, a, b\}$ defined by

$$\begin{aligned} G := [(\perp ab)^*\perp]([\varepsilon] \xrightarrow{a} [\perp ab]) \;\cup\; [(\perp ab)^*(\perp a)^+\perp]([b] \xrightarrow{b} [\varepsilon]) \;\cup\; [(\perp a)^*\perp]([\perp a] \xrightarrow{c} [\varepsilon]) \\ \cup\, \{\iota\} \times \{[\perp]\} \;\cup\; \{o\} \times \{[\perp]\} \end{aligned}$$

We have $L = L(G) \in \mathrm{Rec}^{\ell}_{\mathrm{TrSuffix}^{\mathrm{VAS}}_{\mathrm{det}}}(\mathrm{Vis}^{\mathrm{VAS} \vee \mathrm{Stack}}(\{c\}, \{b\}, \{a\}))$. See Figure 7.

6 Conclusion

Summing up, we have shown how to obtain various Boolean algebras from any family of trace automata closed under level-length synchronization and superposition. The family TrSuffix of trace suffix automata with level-regular contexts and the subfamily $\mathrm{TrSuffix}^{\mathrm{VAS}}$ of vector addition systems satisfy these closure conditions. In particular, we obtain various Boolean algebras of context-sensitive languages accepted by deterministic vector addition systems. However, a better understanding of these languages seems to be a challenging problem. Moreover, it would be interesting to show that some other subfamilies of $\mathrm{TrSuffix}^{\mathrm{VAS}}$ also satisfy the closure conditions stated above.

References

1 R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211, 2004. doi:10.1145/1007352.1007390.

2 D. Caucal. On the Regular Structure of Prefix Rewriting. *Theor. Comput. Sci.*, 106(1):61–86, 1992. doi:10.1016/0304-3975(92)90278-N.

3 D. Caucal. On Infinite Transition Graphs Having a Decidable Monadic Theory. In *Automata, Languages and Programming, 23rd International Colloquium, ICALP96, Paderborn, Germany, 8-12 July 1996, Proceedings*, pages 194–205, 1996. doi:10.1007/3-540-61440-0_128.

4 D. Caucal. On the transition graphs of turing machines. *Theor. Comput. Sci.*, 296(2):195–223, 2003. doi:10.1016/S0304-3975(02)00655-2.

5 D. Caucal. Boolean algebras of unambiguous context-free languages. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India*, pages 83–94, 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1743.

6 D. Caucal and C. Rispal. Recognizability for Automata. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, pages 206–218, 2018. doi:10.1007/978-3-319-98654-8_17.

7 D. Caucal and C. Rispal. Boolean Algebras by Length Recognizability. In Tiziana Margaria, Susanne Graf, and Kim G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, pages 169–185. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-22348-9_11.

8 N. Chomsky. Three models for the description of languages. *IRE Transactions on Information Theory*, 2(3):113–124, September 1956. doi:10.1109/TIT.1956.1056813.

9 S. Crespi-reghizzi and D. Mandrioli. Petri nets and szilard languages. *Information and Control*, 33(2):177–192, 1977. doi:10.1016/S0019-9958(77)90558-7.

10 V. Diekert and G. Rozenberg. *The Book of Traces.* WORLD SCIENTIFIC, 1995. doi:10.1142/2563.

11 S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, Inc., Orlando, FL, USA, 1974.

12 M. Hack. *Decidability questions for Petri nets.* PhD thesis, MIT, Dept. Electrical Engineering, Cambridge, Mass., USA, 1975.

13 B. Hodgson. On Direct Products of Automaton Decidable Theories. *Theor. Comput. Sci.*, 19:331–335, 1982. doi:10.1016/0304-3975(82)90042-1.

14 J. Leroux, V. Penelle, and G. Sutre. On the Context-Freeness Problem for Vector Addition Systems. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 43–52, 2013. doi:10.1109/LICS.2013.9.

15 C. Löding. Model-Checking Infinite Systems Generated by Ground Tree Rewriting. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, pages 280–294, 2002. doi:10.1007/3-540-45931-6_20.

16 A. Mansard. Unfolding of Finite Concurrent Automata. In *Proceedings 11th Interaction and Concurrency Experience, ICE 2018, Madrid, Spain, June 20-21, 2018.*, pages 68–84, 2018. doi:10.4204/EPTCS.279.8.

17 K. Mehlhorn. Pebbling Mountain Ranges and its Application of DCFL-Recognition. In *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, pages 422–435, 1980. doi:10.1007/3-540-10003-2_89.

18 D. Nowotka and J. Srba. Height-Deterministic Pushdown Automata. In *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, pages 125–134, 2007. doi:10.1007/978-3-540-74456-6_13.

19 J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

20 C. Rispal. The synchronized graphs trace the context-sensitive languages. *Electr. Notes Theor. Comput. Sci.*, 68(6):55–70, 2002. doi:10.1016/S1571-0661(04)80533-4.

21 S. R. Schwer. The context-freeness of the languages associated with vector addition systems is decidable. *Theoretical Computer Science*, 98(2):199–247, 1992. doi:10.1016/0304-3975(92)90002-W.

22 R. Valk and G. Vidal-Naquet. Petri nets and regular languages. *Journal of Computer and System Sciences*, 23(3):299–325, 1981. doi:10.1016/0022-0000(81)90067-2.

23 G. Winskel and M. Nielsen. Models for Concurrency. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science (Vol. 4)*, pages 1–148. Oxford University Press, Inc., New York, NY, USA, 1995. URL: http://dl.acm.org/citation.cfm?id=218623.218630.

Widths of Regular and Context-Free Languages

David Mestel
University of Luxembourg
david.mestel@uni.lu

Abstract

Given a partially-ordered finite alphabet Σ and a language $L \subseteq \Sigma^*$, how large can an antichain in L be (where L is given the lexicographic ordering)? More precisely, since L will in general be infinite, we should ask about the rate of growth of maximum antichains consisting of words of length n. This fundamental property of partial orders is known as the width, and in a companion work [10] we show that the problem of computing the information leakage permitted by a deterministic interactive system modeled as a finite-state transducer can be reduced to the problem of computing the width of a certain regular language. In this paper, we show that if L is regular then there is a dichotomy between polynomial and exponential antichain growth. We give a polynomial-time algorithm to distinguish the two cases, and to compute the order of polynomial growth, with the language specified as an NFA. For context-free languages we show that there is a similar dichotomy, but now the problem of distinguishing the two cases is undecidable. Finally, we generalise the lexicographic order to tree languages, and show that for regular tree languages there is a trichotomy between polynomial, exponential and doubly exponential antichain growth.

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

Keywords and phrases Formal languages, combinatorics on words, information flow

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.49

Related Version A full version of the paper is available at `https://arxiv.org/abs/1709.08696`.

Funding The author is supported by FNR under grant number 11689058 (Q-CoDe).

1 Introduction

Computing the size of the largest antichain (set of mutually incomparable elements) is the "central" problem in the extremal combinatorics of partially ordered sets (posets) [14]. In addition to some general theory [7], it has attracted study for a variety of specific sets, beginning with Sperner's Theorem on subsets of $\{1, \ldots, n\}$ ordered by inclusion [12, 2, 11], and for random posets [1]. The size of the largest antichain in a poset is called its *width*.

In this work we study languages L (regular or context-free) over finite partially ordered alphabets, with the lexicographic partial order. Since such languages will in general contain infinite antichains, we study the sets $L_{=n}$ of words of length n, and ask how the width of $L_{=n}$ grows with n; we call this the *antichain growth* rate of L.

In addition to its theoretical interest, the motivation for this work is the study of quantified information flow in computer security: we wish to know whether a pair of isolated agents interacting with a common central system (for example different programs running on a single computer and communicating with the operating system) can obtain any information about each other's actions, and if so how much. In a companion work [10] we show that if the central system is modeled as a deterministic finite-state transducer then this leakage is equivalent to the width of a certain regular language (roughly speaking, antichains corresponding to consistent sets of observations for one agent). The dichotomy we obtain in this paper thus corresponds to a dichotomy between logarithmic and linear information flow.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 49; pp. 49:1–49:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In Section 2 we set out basic definitions and results on the lexicographic order, antichains and antichain growth. In Section 3 we show that for regular languages there is a dichotomy between polynomial and exponential antichain growth, and give a polynomial-time algorithm for distinguishing the two cases. In Section 4 we give a polynomial-time algorithm to compute the order of polynomial antichain growth. In Section 5 we show that for context-free languages there is a similar dichotomy between polynomial and exponential antichain growth, but that the problem of distinguishing the two cases is undecidable. In Section 6 we show that for regular tree languages there is a trichotomy between polynomial, exponential and doubly exponential antichain growth. Finally in Section 7 we discuss open problems.

For reasons of space, many proofs have been omitted or sketched in the conference version of this work; an extended version with full proofs may be found at [9].

2 Languages, lexicographic order and antichains

▶ **Definition 1.** *Let Σ be a finite alphabet equipped with a partial order $\preceq$. Then the lexicographic partial order induced by $\preceq$ on Σ^* is the relation $\preceq$ given by*

(i) *$\epsilon \preceq w$ for all $w \in \Sigma^*$ (where ϵ is the empty word), and*

(ii) *For any $x, y \in \Sigma, w, w' \in \Sigma^*$, we have $xw \preceq yw'$ if and only if either $x \prec y$ or $x = y$ and $w \preceq w'$.*

If words x and y are comparable in this partial order we write $x \sim y$. If x is a prefix of y we write $x \leq y$. For a language L, we will often write $L_{=n}$ to denote the set $\{w \in L \mid |w| = n\}$ (with corresponding definitions for $L_{<n}$, etc.), and $|L|_{=n}$ for $|L_{=n}|$.

The main subject of this work is *antichains*, that is sets of words which are mutually incomparable. It will sometimes be useful also to consider *quasiantichains*, which are sets of words which are incomparable except that the set may include prefixes (note that this is not a standard term). The opposite of an antichain is a *chain*, in which all elements are comparable.

▶ **Definition 2.** *A language L is an* antichain *if for every $l_1, l_2 \in L$ with $l_1 \neq l_2$ we have $l_1 \not\sim l_2$. L is a* quasiantichain *if for every $l_1, l_2 \in L$ we have either $l_1 \leq l_2$, $l_2 \leq l_1$ or $l_1 \not\sim l_2$. L is a* chain *if for all $l_1, l_2 \in L$ we have $l_1 \sim l_2$.*

It is easy to see that the property of being an antichain is preserved by the operations of prefixing, postfixing and concatenation.

▶ **Lemma 3 (Prefixing).** *Let w, w_1, w_2 be any words. Then $w_1 \sim w_2$ if and only if $ww_1 \sim ww_2$. Hence for any language L, wL is an antichain (respectively quasiantichain) if and only if L is an antichain (quasiantichain).*

▶ **Lemma 4 (Postfixing).** *Let w, w_1, w_2 be any words. Then $w_1 \sim w_2$ if $w_1 w \sim w_2 w$. Hence for any language L, Lw is an antichain if L is an antichain.*

▶ **Lemma 5 (Concatenation).** *Let w_1, w_2, w_1', w_2' be any words such that $w_1 \not\leq w_2$ and $w_2 \not\leq w_1$. Then $w_1 w_1' \sim w_2 w_2'$ if and only if $w_1 \sim w_2$. Hence if L_1 and L_2 are antichains then $L_1 L_2$ is an antichain.*

Clearly the property of being an antichain is not preserved by Kleene star, since L^* will contain prefixes for any non-empty L. The best we can hope for is that L^* is a quasiantichain.

▶ **Lemma 6 (Kleene star).** *Let L be an antichain. Then L^* is a quasiantichain.*

Ultimately we are going to care about the size of antichains inside particular languages. Since these will often be unbounded, we choose to ask about the rate of growth; that is, if $L_1, L_2, L_3, \ldots \subseteq L$ are antichains such that L_i consists of words of length i, how quickly can $|L_i|$ grow with i? We will call $\bigcup_i L_i$ an *antichain family* and ask whether it grows exponentially, polynomially, etc.

▶ **Definition 7.** *A language L is an* antichain family *if for each n the set $L_{=n}$ of words in L of length n is an antichain.*

▶ **Definition 8.** *A language L is* exponential *(or* has exponential growth*) if there exists some $\epsilon > 0$ such that*

$$\limsup_{n\to\infty} \frac{|L|_{=n}}{2^{\epsilon n}} > 0,$$

and the supremum of the set of ϵ for which this holds is the order *of exponential growth.*

L is polynomial *(or* has polynomial growth*) if there exists some k such that*

$$\limsup_{n\to\infty} \frac{|L|_{=n}}{n^k} < \infty.$$

If $0 < \limsup_{n\to\infty} \frac{|L|_{=n}}{n^k} < \infty$ then we say that L has polynomial growth of order k.

For notational convenience, we will sometimes later adopt the convention that a language L which is finite (and so $\limsup_{n\to\infty} \frac{|L|_{=n}}{n^k} = 0$ for all k) has polynomial growth of order -1.

A reasonable alternative choice of notation would have been to define the quantity w_n to be the size of the largest antichain consisting of words of length n, and then ask about the growth of the series $w_1, w_2, \ldots$. This is clearly equivalent to the definitions we have given above.

Note that we will sometimes use other characterisations that are clearly equivalent; for instance L has exponential growth if and only if there is some ϵ such that $|L|_{=n} > 2^{\epsilon n}$ infinitely often. We will sometimes refer to a language which is not polynomial as "super-polynomial", or as having "growth beyond all polynomial orders". Of course there exist languages whose growth rates are neither polynomial nor exponential; for instance $|L|_{=n} = \Theta(2^{\sqrt{n}})$.

▶ **Definition 9.** *A language L* has exponential antichain growth *if there is an exponential antichain family $L' \subseteq L$. L* has polynomial antichain growth *if for every antichain family $L' \subseteq L$ we have that L' is polynomial.*

Antichain growth generalises the classical notion of language growth, which is just antichain growth with respect to the discrete partial order (in which all elements of Σ are incomparable).

Note that we could have chosen to define exponential antichain growth as containing an exponential antichain (rather than an exponential antichain family). We will eventually see (Corollary 17) that for regular languages the two notions are equivalent. However, for general languages they are not; indeed the following proposition shows that the two possible definitions are not equivalent even for context-free languages.

▶ **Proposition 10.** *There exists a context-free language L such that L has exponential antichain growth but all antichains in L are finite.*

Proof. Let $\Sigma = \{a, b, 0, 1\}$ with $\prec = \{(a, b)\}$. Let $L = \bigcup_{n=1}^{\infty} L_n = \bigcup_{n=1}^{\infty} a^{n-1}b\{0,1\}^n$. Then each L_n is an antichain of size 2^n consisting of words of length $2n$, but we have $L_1 > L_2 > L_3 > \ldots$ so any antichain is a subset of L_k for some k and hence is finite (the notation $L_1 > L_2$ means that for any $w_1 \in L_1$ and $w_2 \in L_2$ we have $w_2 \prec w_1$). Plainly L is a context-free language. ◀

We observed above that Kleene star does not preserve the property of being an antichain. We conclude this section by establishing Lemma 12, which addresses this problem; if our goal is to find a large antichain, it suffices to find a large quasiantichain (where the precise meaning of "large" is having exponential growth).

As a preliminary, we observe the straightforward fact that taking finite unions does not change the polynomial or exponential growth character of languages.

▶ **Lemma 11.** *Let $L_1, L_2, \ldots, L_k$ be languages, such that $\bigcup_{i=1}^{k} L_i$ has exponential growth of order ϵ (respectively super-polynomial growth). Then L_i has exponential growth of order ϵ (respectively super-polynomial growth) for some i.*

We are now ready to prove Lemma 12.

▶ **Lemma 12.** *Let L be an exponential quasiantichain. Then there exists an exponential antichain $L' \subseteq L$.*

Proof sketch. We construct an exponential subset of L which is prefix-free, and is therefore an exponential antichain. We do this by a Ramsey-style argument: always maintaining the invariant of exponential growth, at each step we pick a fixed word w of length k, throw away that word if it is in the set, and also throw away all longer words of which w is *not* a prefix; by Lemma 11 it is always possible to choose w such that this process preserves the invariant. ◀

3 Regular languages

The dichotomy between polynomial and exponential language growth for regular languages has been independently discovered at least six times (see citations in [4]), in each case based on the fact that a regular language L has polynomial growth if and only if L is *bounded* (that is, $L \subseteq w_1^* \ldots w_k^*$ for some $w_1, \ldots, w_k$); otherwise L has exponential growth.

In [4], Gawrychowski, Krieger, Rampersad and Shallit describe a polynomial time algorithm for determining whether a language is bounded. The key idea is to consider the sets L_q of words which can be generated beginning and ending at state q. L is bounded if and only if for every q we have that L_q is *commutative* (that is, that $L_q \subseteq w^*$ for some w), and this can be checked in polynomial time.

In this section, we generalise this idea to the problem of antichain growth by showing that L has polynomial antichain growth if and only if L_q is a chain for every q, and otherwise L has exponential antichain growth. This is sufficient to establish the dichotomy theorem (Theorem 16). To give an algorithm for distinguishing the two cases (Theorem 18), we show how to produce an automaton whose language is empty if and only if L_q is a chain (roughly speaking the automaton accepts pairs of incomparable words in L_q).

Before proving the main theorems, we first establish (Lemma 13) that if L_1 and L_2 have polynomial antichain growth then so does L_1L_2. Moreover if the rates of polynomial growth of L_1 and L_2 are at most k_1 and k_2 respectively then the rate of polynomial growth of L_1L_2 is at most $k_1 + k_2 + 1$. For the proof of this see the extended version [9].

▶ **Lemma 13.** *Let L_1, L_2 be languages with polynomial antichain growth of order at most k_1 and k_2 respectively. Then L_1L_2 has polynomial antichain growth of order at most $k_1 + k_2 + 1$.*

We are now ready to prove the main theorem, generalising the condition for polynomial language growth (that L_q is commutative for every q) to one for polynomial antichain growth: that L_q is a chain for every relevant q.

▶ **Definition 14.** *A state q of an automaton $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ is* accessible *if q is reachable from q_0 and* co-accessible *if F is reachable from q.*

▶ **Definition 15.** *Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ be an NFA. Then for each $q_1, q_2 \in Q$, the automaton $\mathcal{A}_{q_1,q_2} \triangleq (Q, \Sigma, \Delta, q_1, \{q_2\})$.*

▶ **Theorem 16.** *Let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ be an NFA over a partially ordered alphabet. Then*

(i) *$\mathcal{L}(\mathcal{A})$ has polynomial antichain growth if and only if $\mathcal{L}(\mathcal{A}_{q,q})$ is a chain for every accessible and co-accessible state q, and*

(ii) *if $\mathcal{L}(\mathcal{A})$ does not have polynomial antichain growth then it contains an exponential antichain (and hence has exponential antichain growth).*

Proof. Suppose that $w_1, w_2 \in \mathcal{L}(\mathcal{A}_{q,q})$ with $w_1 \not\sim w_2$ and q accessible and co-accessible, so $w \in \mathcal{L}(\mathcal{A}_{q_0,q})$ and $w' \in \mathcal{L}(\mathcal{A}_{q,q'})$ for some w, w' and some $q' \in F$. Now by the Kleene star Lemma we have that $(w_1 + w_2)^*$ is an exponential quasiantichain and so by Lemma 12 there is an exponential antichain $L' \subseteq (w_1 + w_2)^*$. Then by the Prefixing and Postfixing Lemmas we have that $wL'w' \subseteq L$ is an exponential antichain.

For the converse, we proceed by induction on $|Q|$. Let $Q' = Q \setminus \{q_0\}, F' = F \setminus \{q_0\}$ and $\Delta'(q, a) = \Delta(q, a) \setminus \{q_0\}$ for all $q \in Q', a \in \Sigma$. For any $q \in Q'$, let $\mathcal{A}'_q = (Q', \Sigma, \Delta', q, F')$. Then by the inductive hypothesis we have that $\mathcal{L}(\mathcal{A}'_q)$ has polynomial antichain growth. Also, since $L_{q_0} = \mathcal{L}(\mathcal{A}_{q_0,q_0})$ is a chain it has polynomial (in particular constant) antichain growth. Now we have

$$\mathcal{L}(\mathcal{A}) \subseteq L_{q_0} \cup \bigcup_{q \in Q'} \bigcup_{a \in \Delta(q_0,q)} L_{q_0} a \mathcal{L}(\mathcal{A}'_q).$$

By Lemma 13, each $L_{q_0} a \mathcal{L}(\mathcal{A}'_q)$ also has polynomial antichain growth, and hence by Lemma 11 so does the finite union. ◀

A trivial restatement of part (ii) of the theorem shows that the two possible definitions of antichain growth are equivalent.

▶ **Corollary 17.** *Let L be a regular language. Then L has exponential (respectively superpolynomial) antichain growth if and only if L contains an exponential (respectively superpolynomial) antichain.*

Using Theorem 16 we can produce an algorithm for distinguishing the two cases.

▶ **Theorem 18.** *There exists a polynomial time algorithm to determine whether the language of a given NFA $\mathcal{A}$ has exponential antichain growth.*

Proof sketch. We construct an NFA $\mathcal{B}$ which accepts interleavings of incomparable words over Σ and Σ' (a fresh copy of the alphabet). We then have that $\mathcal{L}(\mathcal{A}_{q,q})$ is a chain if and only if $\mathcal{L}((\mathcal{A}_{q,q} \mathbin{|||} \mathcal{A}'_{q,q}) \cap \mathcal{B})$ is empty, where $\mathcal{A}'$ is a copy of $\mathcal{A}$ over alphabet Σ'. This can be checked in polynomial time. ◀

4 Precise growth rates

In [4] the authors give an algorithm to compute the order of polynomial language growth for the language of a given NFA; on the other hand efficiently computing the order of exponential growth is an open problem. In this section we give an algorithm to compute the order of polynomial antichain growth for the language of a given NFA. We do this by first giving an algorithm for DFA, and then showing that in fact it also works for NFA. We will assume throughout without loss of generality that all states are accessible and co-accessible.

▶ **Definition 19.** *Let $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ be a DFA over a partially ordered alphabet. Let $G_{\mathcal{A}} = (Q, E)$ be the directed graph with vertex-set Q such that $(q, q') \in E$ if and only if $q \xrightarrow{w} q'$ for some $w \in \Sigma^*$.*

Let $G'_{\mathcal{A}} = (Q, E')$ be the directed graph with $(q, q') \in E'$ if and only if there exist words $w \not\sim w' \in \Sigma^*$ such that $q \xrightarrow{w} q$ and $q \xrightarrow{w'} q'$. We will write $L_{q,q'} \triangleq \mathcal{L}(\mathcal{A}_{q,q'})$.

We will generally omit the subscript $\mathcal{A}$s from now on, where this will not cause confusion.

Note that by Theorem 16, we have that G' is a directed acyclic graph (DAG) if and only if $\mathcal{L}(\mathcal{A})$ has polynomial antichain growth. By a similar argument to the proof of Theorem 18, the graph G' can be computed in polynomial time. Clearly G can be computed in polynomial time using a flood fill.

▶ **Definition 20.** *Let $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ be a DFA with polynomial antichain growth. For a directed path $P = q_0 q_1 \ldots q_l$ (not necessarily simple) in $G_{\mathcal{A}}$, let*

$$D(P) = |\{i \in \{0, \ldots, l-1\} | (q_i, q_{i+1}) \in E(G'_{\mathcal{A}})\}| + \begin{cases} 1 \text{ if } |L_{q_m, q_l}| = \infty \\ 0 \text{ otherwise.} \end{cases},$$

where $m = \max\{i + 1 | (q_i, q_{i+1}) \in G'_{\mathcal{A}}\}$ if this exists, and 0 otherwise.

Observe that if $|L_{q_m,q_l}| = \infty$ then we have $ww'^*w'' \subseteq L_{q_m,q_l}$ for some w, w', w''.

▶ **Lemma 21.** *Let $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ be a DFA with polynomial antichain growth. Let $\mathcal{P}$ be the set of directed paths from q_0 to an element of F. Then the quantity*

$$D_{\mathcal{A}} = \max_{P \in \mathcal{P}} D(P)$$

is well-defined and can be computed in polynomial time.

Proof. To show that $D_{\mathcal{A}}$ is well-defined, observe that no directed cycle in G contains an edge in G'. Indeed, suppose that $q_1 q_2 \ldots q_1$ is a directed cycle in G, with $(q_1, q_2) \in E(G')$. Then we have $q_1 \xrightarrow{w} q_1$ and $q_1 \xrightarrow{w'} q_2$ for some $w \not\sim w' \in \Sigma^*$. Also we have $q_2 \xrightarrow{w''} q_1$ for some $w'' \in \Sigma^*$. But then $q_1 \xrightarrow{w'w''} q_1$ and $w'w'' \not\sim w$ by the Concatenation Lemma, contradicting polynomial antichain growth of $\mathcal{L}(\mathcal{A})$. Hence $D(P)$ is bounded.

For a polynomial time algorithm, first expand G and G' by adding a sink vertex v_f for each $f \in F$. For each q such that $|L_{q,f}| = \infty$ put $(q, v_f) \in E(G)$ and $(q, v_f) \in E(G')$. Then add a further vertex v with $(f, v) \in E(G)$ and $(v_f, v) \in E(G)$ for all $f \in F$. Then $D_{\mathcal{A}}$ is precisely the maximum number of edges of G' contained in a directed path from q_0 to v in G.

Form the graph G'' on vertex-set $Q \cup \{v\}$ by $(v_1, v_2) \in E(G'')$ if and only if there is a path from v_1 to v_2 in G containing a single edge of G'. Then we have that G'' is a DAG (by the first observation), and $D_{\mathcal{A}}$ is the longest path from q_0 to v in G'', which can be found by a simple dynamic programming algorithm. ◀

We will show that the order of polynomial antichain growth of $\mathcal{L}(\mathcal{A})$ is precisely $D_{\mathcal{A}} - 1$.

▶ **Lemma 22.** *Let $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ be a DFA with polynomial antichain growth. Then $\mathcal{L}(\mathcal{A})$ has polynomial antichain growth of order at least $D_{\mathcal{A}} - 1$.*

Proof. Let $P = q_0q_1 \dots q_l$ be a path with $D(P) = D_{\mathcal{A}}$. Let $i_1, \dots, i_k$ be such that $(q_{i_j}, q_{i_j+1}) \in E(G'_{\mathcal{A}})$ for all j. Let $w_1, \dots, w_k, w'_1 \dots, w'_k, w \in \Sigma^*$ be such that $w_j \not\sim w'_j$ for all j, $q_{i_j} \xrightarrow{w_j} q_{i_j}$ for all j, $q_{i_j} \xrightarrow{w'_j} q_{i_{j+1}}$ for all $j < k$, $q_{i_k} \xrightarrow{w'_k} q_l$, and $q_0 \xrightarrow{w} q_{i_1}$.

Suppose that $|L_{q_m,q_l}| = \infty$ (with $m = i_k$ defined as in Definition 20), and let $w', w'', w''' \in \Sigma^*$ be such that $w'w''^*w''' \subseteq L_{q_m,q_l}$. Then $L = ww_1^*w'_1w_2^*w'_2 \dots w_k^*w'w''^*w'''$ is an antichain family with polynomial growth of order $k = D_{\mathcal{A}} - 1$. Similarly if $|L_{q_m,q_l}| < \infty$, then $L = ww_1^*w'_1w_2^*w'_2 \dots w_k^*w'_k$ is an antichain with polynomial growth of order $k-1 = D_{\mathcal{A}}-1$. ◀

We will now prove the upper bound. Our strategy will be to classify words by the edges of G' they visit. We first show a preliminary lemma, which bounds the antichain growth from regions between edges of G'.

▶ **Lemma 23.** *Let $q_1, q_2 \in Q$, and let $L \subseteq L_{q_1,q_2}$ be the set of words such that no edges of G' appear in the runs corresponding to elements of L. Then L has antichain growth of order at most 0.*

Proof. Without loss of generality we may assume that $\mathcal{A}$ does not have any transitions labelled by more than a single letter (by introducing additional states if necessary; in particular we can set $Q' = Q \times \Sigma$ and ensure that $\delta'(q, x) \in Q \times \{x\}$ for all $x \in \Sigma$).

We will show that L cannot contain two incomparable words that correspond after removal of loops to the same sets of simple paths in G.[1] Since G is finite and hence contains only finitely many simple paths, this suffices to establish the result.

Suppose that $w_1 \not\sim w_2$ correspond to the same simple path P. Suppose that the first point of divergence of w_1 and w_2 is at state q; that is, that $w_1 = wx_1w'_1$ and $w_2 = wx_2w'_2$ with $x_1 \neq x_2 \in \Sigma$ and $q_1 \xrightarrow{w} q$ (see Figure 1). Without loss of generality we may assume that q and $\delta(q, x_1)$ lie on P.

Since the path for w_2 corresponds to P after removal of cycles, we must have that $w'_2 = w''_2w'''_2$ with $q \xrightarrow{x_2w''_2} q$ and $q \xrightarrow{w'''_2} q_2$. But $w_1 \not\sim w_2$ and $x_1 \neq x_2$ so $x_1 \not\sim x_2$ and so $x_1 \not\sim x_2w''_2$. Hence $(q, \delta(q, x_1)) \in G'$, which is a contradiction. ◀

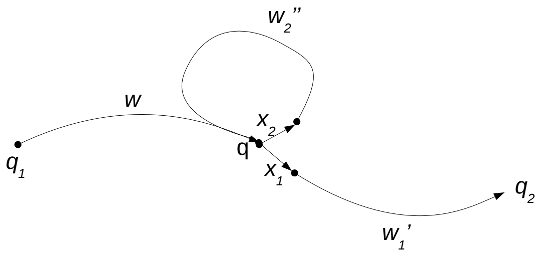


Figure 1 The proof of Lemma 23.

[1] Note that since removal of loops may be done in many different ways, a single path may correspond to multiple simple paths. We are asserting that L cannot contain two incomparable words which correspond to precisely the same *sets* of simple paths.

▶ **Lemma 24.** *Let $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ be a DFA with polynomial antichain growth. Then $\mathcal{L}(\mathcal{A})$ has polynomial antichain growth of order at most $D_{\mathcal{A}} - 1$.*

Proof. We may assume without loss of generality that there is only a single accepting state, say q_f (otherwise consider seperately the automata $\mathcal{A}_1, \ldots, \mathcal{A}_{|F|}$ which agree with $\mathcal{A}$ except for having only a single accepting state; then on the one hand we have $D_{\mathcal{A}} = \max D_{\mathcal{A}_i}$, but on the other hand $\mathcal{L}(\mathcal{A}) = \bigcup \mathcal{A}_i$ which is a finite union and hence the order of antichain growth of $\mathcal{L}(\mathcal{A})$ is the maximum of the orders of growth of the $\mathcal{L}(\mathcal{A}_i)$).

We classify words by the edges of G' that appear in their accepting runs. We shall show that the set of words corresponding to a fixed sequence P of G'-edges has antichain growth of order at most $D(P)$ (where $D(P) = |P| - 1$ or $|P|$ depending on whether the set of accepted words beginning at the last vertex of P is finite). Since the number of relevant G'-edge sequences is finite (recalling that no edge of G' is contained in a directed cycle in G and so no G'-edge can appear more than once), this will suffice to establish the result.

Let $(q_1, q_1'), \ldots, (q_k, q_k')$ be a set of G'-edges. Then the set L of words which have this sequence of G'-edges in their run is given by

$$L = L'_{q_0,q_1} X_1 L'_{q_1',q_2} X_2 L'_{q_2',q_3} \ldots X_k L'_{q_k',q_f},$$

where $X_i = \{x \in \Sigma \mid \delta(q_i, x) = q_i'\}$ and $L'_{q,q'} \subset L_{q,q'}$ is the set of words whose runs do not include edges of G'.

The X_i are finite and hence have antichain growth of order -1. By Lemma 23 the $L'_{q_i',q_{i+1}}$ and also L'_{q_0,q_1} and L'_{q_k',q_f} have antichain growth of order at most 0. Moreover if L_{q_k',q_f} is finite then so is $L'_{q_k',q_f} \subseteq L_{q_k',q_f}$ and so it has antichain growth of order -1. The result follows by Lemma 13. ◀

Combining Lemmas 21, 22 and 24 yields

▶ **Theorem 25.** *Let $\mathcal{A} = (Q, q_0, F, \Sigma, \delta)$ be a DFA with polynomial antichain growth. Then $\mathcal{L}(\mathcal{A})$ has polynomial antichain growth of order exactly $D_{\mathcal{A}} - 1$, which can be computed in polynomial time.*

We now show how to extend this algorithm to the case of NFA. Note that $D_{\mathcal{A}}$ as defined above is well-defined for NFA just as for DFA, and that the algorithm to compute it in polynomial time is equally applicable. It therefore remains to show that for NFA we also have that if $\mathcal{A}$ has polynomial antichain growth then it has antichain growth of order exactly $D_{\mathcal{A}} - 1$.

We do this by showing (Lemma 27) that $D_{\mathcal{A}}$ depends only on the language $\mathcal{L}(\mathcal{A})$, so that if $\mathcal{A}$ and $\mathcal{A}'$ are NFA with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ then $D_{\mathcal{A}} = D_{\mathcal{A}'}$. Having shown this we then consider $\mathcal{A}'$ to be the determinisation of $\mathcal{A}$. This is a DFA with $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$, and by Theorem 25 we have that $\mathcal{L}(\mathcal{A}')$ has polynomial antichain growth of order $D_{\mathcal{A}'} - 1 = D_{\mathcal{A}} - 1$.

We will first show (Lemma 26) that if $L = v_0 w_1^* v_1 w_2^* v_2 \ldots w_k^* v_k \subseteq \mathcal{L}(\mathcal{A})$ then there exists a single sequence of states $q_1, q_2, \ldots, q_k$ which essentially realises L (that is, up to various offsets we have $v_i \in \mathcal{L}(\mathcal{A}_{q_i,q_{i+1}})$ and $w_i^* \in \mathcal{L}(\mathcal{A}_{q_i,q_i})$).

▶ **Lemma 26.** *Let $\mathcal{A} = (Q, q_0, F, \Sigma, \Delta)$ be an NFA such that $v_0 w_1^* v_1 w_2^* v_2 \ldots w_k^* v_k \subseteq \mathcal{L}(\mathcal{A})$. Then then there exists a sequence of states $q_1, q_2, \ldots, q_{k+1}$ and integers $m_1, m_2, \ldots m_k$, $m_1', m_2', \ldots, m_k'$ and $n_1, n_2, \ldots, n_k$ such that*

(i) *$v_0 w_1^{m_1} \in \mathcal{L}(\mathcal{A}_{q_0,q_1})$ and $w_k^{m_k'} v_k \in \mathcal{L}(\mathcal{A}_{q_k,F})$,*

(ii) *for all $0 < i < k$ we have $w_i^{m_i'} v_i w_{i+1}^{m_{i+1}} \in \mathcal{L}(\mathcal{A}_{q_i,q_{i+1}})$, and*

(iii) *for all $0 < i \leq k$ we have $w_i^{n_i} \in \mathcal{L}(\mathcal{A}_{q_i,q_i})$.*

Proof. Consider an accepting run for $v_0w_1^{|Q|+1}v_1w_2^{|Q|+1}v_2\dots w_k^{|Q|+1}v_k \in \mathcal{L}(\mathcal{A})$, and write $q(s)$ for the state reached in this run after the word s. By the pigeon-hole principle, we must have $q(v_0w_1^{m_1}) = q(v_0w^{m_1+n_1}) = q_1$ (say) for some $m_1 \geq 0$ and some $n_1 > 0$ with $m_1 + n_1 \leq |Q| + 1$. Let $m'_1 = |Q| + 1 - m_1 - n_1$. Similarly for each i we have $q(v_1w_1^{|Q|+1}v_2\dots w_i^{m_i}) = q(v_1w_1^{|Q|+1}v_2\dots w_i^{m_i+n_i}) = q_i$ (say) for some $m_i \geq 0$ and $n_i > 0$ with $m_i + n_i \leq |Q| + 1$. Let $m'_i = |Q| + 1 - m_i - n_i$. Then these q_i, m_i, m'_i and n_i give the result. ◀

▶ **Lemma 27.** *Let $\mathcal{A}$ and $\mathcal{A}'$ be NFA with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Then $D_{\mathcal{A}} = D_{\mathcal{A}'}$.*

Proof. Let $\mathcal{A} = (Q, q_0, F, \Sigma, \Delta)$ and $\mathcal{A}' = (Q', q'_0, F', \Sigma, \Delta')$.

Suppose that $D_{\mathcal{A}'} = k$. Then by an identical argument to the proof of Lemma 22 we have that $v_0w_1^*v_1w_2^*v_2\dots w_k^*v_k \subseteq \mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ for some $v_0, \dots, v_k, w_1, \dots, w_k \in \Sigma^*$ with $w_i \not\sim v_i$. Then by Lemma 26 there exists a sequence of states $q_1, q_2, \dots, q_{k+1} \in Q$ and integers $m_1, m_2, \dots, m_k, m'_1, m'_2, \dots m'_k$ and $n_1, n_2, \dots, n_k$ such that (i)–(iii) in the statement of the lemma hold. Now since $w_i \not\sim v_i$ we have $w_i^{k_in_i} \not\sim w_i^{m'_i}v_iw_{i+1}^{m_{i+1}}$ for sufficiently large k_i and so $D_{\mathcal{A}} \geq k = D_{\mathcal{A}'}$. Similarly $D_{\mathcal{A}'} \geq D_{\mathcal{A}}$, and hence $D_{\mathcal{A}} = D_{\mathcal{A}'}$. ◀

▶ **Theorem 28.** *Let $\mathcal{A}$ be an NFA with polynomial antichain growth. Then $\mathcal{L}(\mathcal{A})$ has polynomial antichain growth of order exactly $D_{\mathcal{A}} - 1$.*

Proof. Let $\mathcal{A}'$ be the powerset determinisation of $\mathcal{A}$, so $\mathcal{A}'$ is a DFA with $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. By Theorem 25, $\mathcal{L}(\mathcal{A}')$ has polynomial antichain growth of order exactly $D_{\mathcal{A}'} - 1$, and by Lemma 27 we have $D_{\mathcal{A}'} = D_{\mathcal{A}}$. ◀

5 Context-free languages

In [6], Ginsburg and Spanier show (Theorem 5.1) that a context-free grammar G generates a bounded language if and only if the sets $L_A(G)$ and $R_A(G)$ are commutative for all nonterminals A, where L_A and R_A are respectively the sets of possible w and u in productions $A \overset{*}{\Rightarrow} wAu$. They also give an algorithm to decide this (which [4] improves to be in polynomial time).

We generalise this to our problem by showing that G generates a language with polynomial antichain growth if and only $L_A(G)$ and also the sets $R_{A,w}(G)$ of possible u for each fixed w are chains, and that otherwise $\mathcal{L}(G)$ has exponential antichain growth. However, we will show that the problem of distinguishing the two cases is undecidable, by reduction from the CFG intersection emptiness problem.

Except where otherwise specified, we will assume all CFGs have starting symbol S and that all nonterminals are accessible and co-accessible: for any nonterminal A we have $S \overset{*}{\Rightarrow} uAu'$ for some $u, u' \in \Sigma^*$ and $A \overset{*}{\Rightarrow} v$ for some $v \in \Sigma^*$.

▶ **Definition 29.** *Let G be a context-free grammar (CFG) over Σ. Then for any nonterminal A let*

$$L_A(G) = \{w \in \Sigma^* \mid \exists\, u \in \Sigma^* : A \overset{*}{\Rightarrow} wAu\}.$$

▶ **Lemma 30.** *Let G be a CFG over Σ and A some nonterminal such that $L_A(G)$ is not a chain. Then $\mathcal{L}(G)$ contains an exponential antichain.*

Proof. Since $L_A(G)$ is not a chain, we have w_1, w_2, u_1, u_2 with $w_1 \not\sim w_2$ such that $A \overset{*}{\Rightarrow} w_1Au_1$ and $A \overset{*}{\Rightarrow} w_2Au_2$. Now A is accessible and co-accessible so also $S \overset{*}{\Rightarrow} uAu'$ and $A \overset{*}{\Rightarrow} v$ for some $u, u', v \in \Sigma^*$.

Hence $uw_{i_1}w_{i_2}\dots w_{i_k}vu_{i_k}u_{i_{k-1}}\dots u_{i_1}u' \subseteq \mathcal{L}(G)$, for any $i_1i_2\dots i_k \in \{1,2\}^*$. Write $\phi : (w_1+w_2)^* \to (u_1+u_2)^*$ for the map $w_{i_1}w_{i_2}\dots w_{i_k} \mapsto u_{i_k}u_{i_{k-1}}\dots u_{i_1}$ (with any ambiguity resolved arbitrarily).

Now $\{w_{i_1}w_{i_2}\dots w_{i_k}|i_1\dots i_k \in \{1,2\}^*\} = (w_1+w_2)^*$ is a quasiantichain by Lemma 6, clearly it is exponential and hence by Lemma 12 it contains an exponential antichain L. By the Concatenation Lemma we have that $L' = \{lv\phi(l)|l \in L\}$ is an antichain, and it is exponential because there is a bijection between L and L' such that the length of each word in L' exceeds the length of the corresponding word in L by a factor of at most $\frac{|v|+\max(|u_1|,|u_2|)}{\min(|w_1|,|w_2|)}$. By the Prefixing and Postfixing Lemmas we have that $uL'u' \subseteq \mathcal{L}(G)$ is an exponential antichain. ◀

▶ **Definition 31.** *Let G be a CFG over Σ. Then for any nonterminal A and any $w \in \Sigma^*$, let*

$$R_{A,w}(G) = \{u \in \Sigma^*|A \overset{*}{\Rightarrow} wAu\}.$$

▶ **Lemma 32.** *Let G be a CFG over Σ, A some nonterminal and $w \in \Sigma^*$ such that $R_{A,w}(G)$ is not a chain. Then $\mathcal{L}(G)$ has exponential antichain growth.*

Proof. We have $v, w, u, u' \in \Sigma^*$ and $u_1 \not\sim u_2 \in \Sigma^*$ such that $S \overset{*}{\Rightarrow} uAu'$, $A \overset{*}{\Rightarrow} v$, $A \overset{*}{\Rightarrow} wAu_1$ and $A \overset{*}{\Rightarrow} wAu_2$. Let $L_i = uw^{2i}v(u_1u_2+u_2u_1)^iu'$. Then L_i is an antichain and $\bigcup_{i=1}^{\infty} L_i$ is an exponential antichain family. ◀

▶ **Lemma 33.** *Let G be a CFG over Σ such that $L_A(G)$ and $R_{A,w}(G)$ are chains for all nonterminals A and all $w \in \Sigma^*$. Then $\mathcal{L}(G)$ has polynomial antichain growth.*

Proof sketch. Induction on the number of nonterminals, similarly to the proof of Theorem 16. ◀

Combining these three lemmas gives:

▶ **Theorem 34.** *Let L be a context-free language. Then either L has exponential antichain growth or L has polynomial antichain growth.*

It is a straightforward exercise to show that the ambiguity of an NFA (the maximum number of accepting paths corresponding to a given word) can be represented as the width of a suitable context-free language, and hence Theorem 34 implies the well-known result that the ambiguity of an NFA has either polynomial or exponential growth (see Theorem 4.1 of [13]).

We now show that the problem of distinguishing the two cases of antichain growth is undecidable for context-free languages, by reduction from the CFG intersection emptiness problem. In fact, it is undecidable even to determine whether a given CFG generates a chain.

▶ **Definition 35.** CFG-Intersection *is the problem of determining whether two given CFGs have non-empty intersection.* CFG-Chain *is the problem of determining whether the language generated by a given CFG is a chain.* CFG-ExpAntichain *is the problem of determining whether the language generated by a given CFG has exponential antichain growth.*

▶ **Lemma 36.** CFG-Intersection *is undecidable.*

Proof. [5], Theorem 4.2.1. ◀

▶ **Lemma 37.** *There is a polynomial time reduction from* CFG-Intersection *to* CFG-Chain.

Proof. Let G_1, G_2 be arbitrary CFGs over alphabet Σ. Let $\widetilde{\Sigma} = \Sigma \cup \{0, 1\}$, with an arbitrary linear order on Σ, and $\Sigma < 0, \Sigma < 1$ but 0 and 1 incomparable. Let $\widetilde{G}$ be a CFG such that

$$\mathcal{L}(\widetilde{G}) = (\mathcal{L}(G_1)0) \cup (\mathcal{L}(G_2)1)$$

(which can trivially be constructed with polynomial blowup). Then $\mathcal{L}(\widetilde{G})$ is a chain if and only if $G_1 \cap G_2 = \emptyset$. ◀

▶ **Lemma 38.** *Let L be a prefix-free chain. Then L^* is a chain.*

Proof. Let $lw \not\sim l'w'$ be a minimum-length counterexample with $l, l' \in L$ and $w, w' \in L^*$. By minimality and the Prefixing Lemma we have that $l \neq l'$. Then by the Concatenation Lemma since L is prefix-free we have that $l \not\sim l'$, which is a contradiction. ◀

▶ **Lemma 39.** *There is a polynomial time reduction from* CFG-Chain *to* CFG-ExpAntichain.

Proof. Let G be a CFG over a partially ordered alphabet Σ. Let $\widetilde{\Sigma} = \Sigma \cup \{0\}$, with $\Sigma < 0$. Let $\widetilde{G}$ be a CFG such that $\mathcal{L}(\widetilde{G}) = (\mathcal{L}(G)0)^*$.

We claim that $\mathcal{L}(\widetilde{G})$ has exponential antichain growth if and only if $\mathcal{L}(G)$ is not a chain. Indeed, suppose that $l_1 \not\sim l_2 \in \mathcal{L}(G)$. Then $l_1 0 \not\sim l_2 0$ and so by Lemmas 6 and 12 we have that $(l_1 0 + l_2 0)^* \subseteq \mathcal{L}(\widetilde{G})$ contains an exponential antichain.

Conversely, suppose that $\mathcal{L}(G)$ is a chain. Then $\mathcal{L}(G)0$ is a prefix-free chain and so by Lemma 38 we have that $\mathcal{L}(\widetilde{G})$ is a chain. ◀

Combining these lemmas gives:

▶ **Theorem 40.** *The problems* CFG-Chain *and* CFG-ExpAntichain *are undecidable.*

6 Tree automata

In this section, we generalise the definition of the lexicographic ordering to tree languages, and prove a trichotomy theorem: regular tree languages have antichain growth which is either polynomial, exponential or doubly exponential.

Notation and definitions (other than for the lexicographic ordering) are taken from [3], to which the reader is referred for a more detailed treatment. Results in this section are stated without proof; all proofs may be found in the extended version [9].

▶ **Definition 41.** *Let $\mathcal{F}$ be a finite set of function symbols of arity ≥ 0, and $\mathcal{X}$ a set of variables. Write $\mathcal{F}_p$ for the set of function symbols of arity p. Let $T(\mathcal{F}, \mathcal{X})$ be the set of terms over $\mathcal{F}$ and $\mathcal{X}$. Let $T(\mathcal{F})$ be the set of* ground terms *over $\mathcal{F}$, which is also the set of* ranked ordered trees *labelled by $\mathcal{F}$ (with rank given by arity as function symbols).*

For example, the set of ordered binary trees is $T(\mathcal{F})$, where $\mathcal{F} = \{f, g, c\}$ and f has arity 2, g arity 1 and c arity 0.

Note that this generalises the definition of finite words over an alphabet Σ, by taking $\mathcal{F} = \Sigma \cup \{\epsilon\}$, giving each $a \in \Sigma$ arity one and ϵ arity zero.

A term t is *linear* if no free variable appears more than once in t. A linear term mentioning k free variables is a *k-ary context.*

▶ **Definition 42.** *Let $\mathcal{F}$ be equipped with a partial order $\preceq$. Then the lexicographic partial order induced by $\preceq$ on $T(\mathcal{F})$ is the relation $\preceq$ defined as follows: for any $f \in \mathcal{F}_p, f' \in \mathcal{F}_q$ and any $t_1, \ldots, t_p \in T(\mathcal{F})$ and $t'_1, \ldots, t'_q \in T(\mathcal{F})$ we have $f(t_1, \ldots, t_p) \preceq f'(t'_1, \ldots, t'_q)$ if and only if either $f \prec f'$ or $f = f'$ and $t_i \preceq t'_i$ for all i.*

Note that this generalises Definition 1, by taking $\epsilon \preceq a$ for all $a \in \Sigma$. As before we will write $t \sim t'$ if $t, t' \in T(\mathcal{F})$ are related by the lexicographic order; the definitions of *chain* and *antichain* are as before. To quantify antichain growth we need a notion of the size of a tree. The measure we will use will be *height*:

▶ **Definition 43.** *The* height *function $h : T(\mathcal{F}, \mathcal{X}) \to \mathbb{N}$ is defined by $h(x) = 0$ for all $x \in \mathcal{X}$, $h(t) = 1$ for all $t \in \mathcal{F}_0$ and $h(t(t_1, \ldots, t_n)) = 1 + \max(h(t_1, \ldots, t_n))$ for all $t \in \mathcal{F}_n$ $(n \geq 1)$ and $t_1, \ldots, t_n \in T(\mathcal{F}, \mathcal{X})$. For a language L, the set $\{t \in L \mid h(t) = k\}$ is denoted $L_{=k}$.*

For example, taking the earlier example of binary trees, ground terms of height 3 include $f(f(c, c), f(c, c))$, $f(c, f(c, c))$ and $g(f(c, c))$.

We say that L has *doubly exponential* antichain growth if there is some ϵ such that the maximum size antichain in $L_{=n}$ exceeds $2^{2^{\epsilon n}}$ infinitely often.

▶ **Definition 44.** *A* nondeterministic finite tree automaton *(NFTA) over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ where Q is a set of unary states, $Q_f \subseteq Q$ is a set of final states, and Δ a set of transition rules of type $f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n))$, for $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$ and $x_1, \ldots, x_n \in \mathcal{X}$. The* move relation $\underset{\mathcal{A}}{\to}$ *is defined by applying a transition rule possibly inside a context and possibly with substitutions for the x_i. The reflexive transitive closure of $\underset{\mathcal{A}}{\to}$ is denoted $\overset{*}{\underset{\mathcal{A}}{\to}}$.*

A tree $t \in T(\mathcal{F})$ is accepted *by $\mathcal{A}$ if there is some $q \in Q_f$ such that $t \overset{*}{\underset{\mathcal{A}}{\to}} q(t)$. The set of trees accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$.*

Again this generalises the definition of an NFA: put in transitions $\epsilon \to q(\epsilon)$ for all accepting states q, $a(q(x)) \to q'(a(x))$ whenever $q \in \Delta(q', a)$, and set Q_f as the initial state.

The critical idea for the proof is to find the appropriate analogue of L_q. This turns out to be the set P_q of binary contexts such that if the free variables are assigned state q then the root can also be given state q. By analogy to the “trousers decomposition” of differential geometry (also known as the “pants decomposition”), we refer to such a context as a *pair of trousers*. It turns out that a sufficient condition for L to have doubly exponential antichain growth is for P_q to be non-empty for some q (note that this does not depend on the particular partial order on Σ). On the other hand, if P_q is empty for all q, then there is in a suitable sense no branching and so we have a similar situation to ordinary languages.

▶ **Definition 45.** *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be an NFTA and $q \in Q$. A linear term $t \in T(\mathcal{F}, \{x_1, x_2\})$ is a* pair of trousers *with respect to q if x_1, x_2 appear in t and $t[x_1 \leftarrow q(x_1), x_2 \leftarrow q(x_2)] \overset{*}{\underset{\mathcal{A}}{\to}} q(t)$. The set of pairs of trousers with respect to q is denoted $P_q(\mathcal{A})$.*

▶ **Lemma 46.** *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a reduced NFTA. If there exists some $q \in Q$ such that $P_q(\mathcal{A})$ is non-empty, then $\mathcal{L}(\mathcal{A})$ contains a doubly exponential antichain.*

▶ **Lemma 47.** *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a reduced NFTA such that $P_q(\mathcal{A}) = \emptyset$ for all $q \in Q$. Then $\mathcal{L}(\mathcal{A})$ has at most exponential growth.*

In the case where there are no pairs of trousers, the situation is essentially equivalent to ordinary NFA, and so we have a further dichotomy between exponential and polynomial antichain growth. To show this, we define a set equivalent to $L_{q,q}$, and show that we have polynomial growth if it is a chain and exponential growth otherwise.

▶ **Definition 48.** *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be an NFTA, and $q \in Q$. Define $\mathcal{L}_q(\mathcal{A}) \subseteq T(\mathcal{F}, \{x_1\})$ to be the set of unary contexts t such that $t[x_1 \leftarrow q(x_1)] \overset{*}{\underset{\mathcal{A}}{\to}} q(t)$.*

Note that unary contexts are linear terms in which *exactly* one free variable appears, so $\mathcal{L}_q(\mathcal{A})$ does not contain ground terms. Note also that $x_1 \in \mathcal{L}_q(\mathcal{A})$ for any $\mathcal{A}$.

To give meaning to the statement "$\mathcal{L}_q(\mathcal{A})$ is a chain", we must extend the definition of the lexicographic order from the set $T(\mathcal{F})$ of ground terms to the set $T(\mathcal{F}, \{x_1\})$ of unary contexts. We do this by extending the relation $\preceq$ on $\mathcal{F}$ to $\mathcal{F} \cup \{x_1\}$ by $x_1 \preceq f$ for all $f \in \mathcal{F}$, and extending this to the lexicographic order as before.

▶ **Lemma 49.** *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a reduced NFTA such that $P_q(\mathcal{A}) = \emptyset$ for all q. Then $\mathcal{L}(\mathcal{A})$ has polynomial antichain growth if $\mathcal{L}_q(\mathcal{A})$ is a chain for all q, and otherwise $\mathcal{L}(\mathcal{A})$ has exponential antichain growth.*

Combining these lemmas gives

▶ **Theorem 50.** *Let L be a regular tree language over a partially ordered alphabet. Then L has either doubly exponential antichain growth, singly exponential antichain growth, or polynomial antichain growth.*

The special case of the trivial partial order (in which elements are only comparable to themselves) yields the fact that the language growth of any regular tree language is either polynomial, exponential or doubly exponential, which may not have previously appeared in the literature.

▶ **Corollary 51.** *Let L be a regular tree language. Then L has either doubly exponential language growth, singly exponential language growth or polynomial language growth.*

Finally, we show that there is a polynomial algorithm to detect doubly exponential growth, by determining whether or not the language of a given NFTA contains a pair of trousers.

▶ **Theorem 52.** *There exists a polynomial time algorithm to determine whether the language of a given NFTA has doubly exponential growth.*

7 Open problems

It is remarkable that, many decades after the discovery of the dichotomy between polynomial and exponential language growth, and 11 years after the work of Gawrychowski, Krieger, Rampersad and Shallit [4], it remains unknown whether there is an efficient algorithm to compute the order of exponential language growth of a given NFA. Consequently we consider that resolving this question (by providing either a polynomial-time algorithm or an appropriate hardness result) is the most important open problem in this area.

For a DFA, on the other hand, the order of exponential language growth is easily computed as the spectral radius of the transition matrix. However, it is not clear how such "algebraic" methods can be applied to the case of antichain growth, and so a second open problem is to find a polynomial-time algorithm to compute the order of exponential antichain growth for DFA. Such a result would have immediate application to the field of quantified information flow, since it would allow one to compute the flow rate in the "dangerous" linear case, at the cost of determinising the automaton representing the system (with overhead roughly corresponding to the amount of hidden state the system contains).

The final problem in this direction is the combination of the preceding two: to find a polynomial-time algorithm to compute the order of exponential antichain growth for a given NFA.

Alternatively we may wish to ask not about growth rates in the asymptotic limit, but instead about the precise width of $L_{=n}$ or $L_{\leq n}$ for given n. This is particularly relevant to applications in computer security, where we may want not just an approximation "for sufficiently large n" but a concrete guarantee. For the case of a language given as a DFA and n given in unary there is a straightforward dynamic programming algorithm to compute these quantities (for details see p.89 of [8]), but what about for NFA and for more concise representations of n?

Finally we pose a more speculative question: what other phenomena, apart from information flow, can antichains with respect to the lexicographic order usefully represent?

References

1 Graham Brightwell. Random k-dimensional orders: Width and number of linear extensions. *Order*, 9(4):333–342, 1992.

2 E. Rodney Canfield. On a problem of Rota. *Advances in Mathematics*, 29(1):1–10, 1978.

3 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

4 Paweł Gawrychowski, Dalia Krieger, Narad Rampersad, and Jeffrey Shallit. Finding the growth rate of a regular of [sic] context-free language in polynomial time. In *Developments in Language Theory*, pages 339–358. Springer, 2008.

5 Seymour Ginsburg. *The Mathematical Theory of Context Free Languages.* McGraw-Hill Book Company, 1966.

6 Seymour Ginsburg and Edwin H. Spanier. Bounded ALGOL-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.

7 D. Kleitman, M. Edelberg, and D. Lubell. Maximal sized antichains in partial orders. *Discrete Mathematics*, 1(1):47–53, 1971.

8 David Mestel. *Quantifying information flow.* PhD thesis, University of Oxford, 2018.

9 David Mestel. Widths of regular and context-free languages. *CoRR*, abs/1709.08696, 2018. `arXiv:1709.08696`.

10 David Mestel. Quantifying information flow in interactive systems. In *Proc. 32nd IEEE Computer Security Foundations Symposium (CSF '19)*, pages 414–427, June 2019.

11 G. W. Peck. Maximum antichains of rectangular arrays. *Journal of Combinatorial Theory, Series A*, 27(3):397–400, 1979.

12 Emanuel Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928.

13 Andreas Weber and Helmut Seidl. On the Degree of Ambiguity of Finite Automata. *Theor. Comput. Sci.*, 88(2):325–349, October 1991.

14 Douglas B. West. Extremal Problems in Partially Ordered Sets. In Ivan Rival, editor, *Ordered Sets: Proceedings of the NATO Advanced Study Institute held at Banff, Canada, August 28 to September 12, 1981*, pages 473–521. Springer Netherlands, Dordrecht, 1982.

Degrees of Ambiguity of Büchi Tree Automata

Alexander Rabinovich
Tel Aviv University, Israel
https://www.cs.tau.ac.il/~rabinoa/
rabinoa@tauex.tau.ac.il

Doron Tiferet[1]
Tel Aviv University, Israel
sdoron5.t2@gmail.com

Abstract

An automaton is unambiguous if for every input it has at most one accepting computation. An automaton is finitely (respectively, countably) ambiguous if for every input it has at most finitely (respectively, countably) many accepting computations. An automaton is boundedly ambiguous if there is $k \in \mathbb{N}$, such that for every input it has at most k accepting computations. We consider nondeterministic Büchi automata (NBA) over infinite trees and prove that it is decidable in polynomial time, whether an automaton is unambiguous, boundedly ambiguous, finitely ambiguous, or countably ambiguous.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases automata on infinite trees, ambiguous automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.50

Funding Supported in part by Len Blavatnik and the Blavatnik Family foundation.

1 Introduction

Degrees of Ambiguity

The relationship between deterministic and nondeterministic machines plays a central role in computer science. An important topic is a comparison of expressiveness, succinctness and complexity of deterministic and nondeterministic models. Various restricted forms of nondeterminism were suggested and investigated (see [4, 5] for recent surveys).

Probably, the oldest restricted form of nondeterminism is unambiguity. An automaton is unambiguous if for every input there is at most one accepting run. For automata over finite words there is a rich and well-developed theory on the relationship between deterministic, unambiguous and nondeterministic automata [5]. All three models have the same expressive power. Unambiguous automata are exponentially more succinct than deterministic ones, and nondeterministic automata are exponentially more succinct than unambiguous ones [6, 7].

Many other notions of ambiguity were suggested and investigated. A recent paper [5] surveys works on the degree of ambiguity and on various nondeterminism measures for finite automata on words.

An automaton is k-*ambiguous* if on every input it has at most k accepting runs; it is *boundedly ambiguous* if it is k-ambiguous for some k; it is *finitely ambiguous* if on every input it has finitely many accepting runs.

It is clear that an unambiguous automaton is k-ambiguous for every $k > 0$, and a k-ambiguous automaton is finitely ambiguous. The reverse implications fail. For ϵ-free automata over words (and over finite trees), on every input there are at most finitely many

[1] corresponding author

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 50; pp. 50:1–50:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

accepting runs. Hence, every ϵ-free automaton on finite words and on finite trees is finitely ambiguous. However, over ω-words there are nondeterministic automata with uncountably many accepting runs. Over ω-words and over infinite trees, finitely ambiguous automata are a proper subclass of the class of countably ambiguous automata, which is a proper subclass of nondeterministic automata. Our main result is:

▶ **Theorem 1.** *There are polynomial time algorithms that decide whether a Büchi automaton over trees is unambiguous, boundedly ambiguous, finitely ambiguous, or countably ambiguous.*

Over infinite trees, Büchi tree automata are less expressive than Monadic Second-Order Logic or parity automata. In Sect. 8 we will show that the problem whether a parity tree automaton is ambiguous is co-NP complete.

Related Works

Weber and Seidl [14] investigated several classes of ambiguous automata on words and obtained polynomial time algorithms for deciding the membership in each of these classes. Their algorithms were derived from structural characterizations of the classes.

In particular, they proved that the following Bounded Ambiguity Criterion (BA) characterizes whether there is a bound k such that a nondeterministic automaton on words has at most k accepting runs on each word.

Forbidden Pattern for Bounded Ambiguity: There are distinct useful[2] states $p, q \in Q$ such that for some word u, there are runs on u from p to p, from p to q and from q to q.

Weber and Seidl [14] proved that an NFA is not boundedly ambiguous iff it contains the forbidden pattern for bounded ambiguity. This pattern is testable in polynomial time; hence, it can be decided in polynomial time whether the degree of ambiguity of a NFA is bounded.

Seidl [12] provided a structural characterization of bounded ambiguity for automata on finite trees and derived a polynomial algorithm to decide whether such an automaton is boundedly ambiguous.

Löding and Pirogov [8] and Rabinovich [10] provided structural characterizations and polynomial algorithms for bounded, finite and countable ambiguity of Büchi automata on ω-words. These characterizations and algorithms can be adopted for other acceptance conditions: parity, Rabin, Muller, etc.

Our proof of Theorem 1 will first provide structural characterizations of bounded, finite and countable ambiguity of automata on infinite trees, and then derive polynomial algorithms.

As far as we know, the degrees of ambiguity for automata over infinite trees have not been investigated. The decidability whether an automaton on infinite trees is finitely ambiguous or countably ambiguous can be obtained from the results of Bárány et al. in [2], where an extension of monadic second-order logic of order with the cardinality quantifiers "there exist uncountably many sets," "there are countably many sets," "there are finitely many sets" (MSO($\exists^{<\aleph_0}, \exists^{>\aleph_0}$)) was investigated. It was proved that, over the class of finitely branching trees, MSO($\exists^{<\aleph_0}, \exists^{>\aleph_0}$) is (effectively) equally expressive to plain monadic second-order logic of order (MSO). It is a routine exercise for a given automaton on infinite trees to write sentences in MSO($\exists^{<\aleph_0}, \exists^{>\aleph_0}$) that express "the automaton has finitely many accepting runs," "the automaton has countably many accepting runs," and "the automaton has uncountably many accepting runs." By combining these with Rabin's theorem on decidability of MSO over infinite trees we obtain that it is decidable whether an automaton is finitely or countably ambiguous. Unfortunately, the complexity of the algorithm extracted from this proof is (at least) triple exponential. Our proofs are inspired by techniques used in [2].

[2] A state is useful if it is on an accepting run.

Organization of the paper. The next section contains standard definitions and notations about tree automata. The main results are stated in Sect. 3 and are proved in Sects. 4–7. The last section presents the conclusion and further results. Missing proofs will appear in the full paper.

2 Preliminaries

We recall here standard terminology and notations about trees and automata [13, 9].

Trees. A **tree order** $\leq$ on a set t is a partial order with a unique minimal element (the root of t) such that for every $u \in t$, the set $\{v \mid v \leq u\}$ is finite and linearly ordered by $\leq$. We use standard terminology and notations: u is an *ancestor* of v if $u \leq v$, u is a *child* of v, u is a *leaf*, u and v are incomparable - denoted by $u \perp v$ - if neither $u \leq v$ nor $v \leq u$; a subset A of t is an antichain, if its elements are incomparable with each other.

If $\leq$ is a tree order over t and $u \in t$, we denote by $t_{\geq u}$ the restriction of $\leq$ to the set $\{v \mid v \geq u\}$; $t_{\geq u}$ is called the subtree of t rooted at u. A binary tree is a tree order with a partition of children into two sets - left/right child such that every non-leaf node has exactly one left child and one right child. The full binary tree is a binary tree without leaves.

There is a natural order isomorphism between the full binary trees and the set of strings $\{l, r\}^*$ with prefix order; it maps ϵ to the root, l to the left child of root, etc. We will often refer to a node in the full binary tree by the corresponding string over $\{l, r\}$.

If Σ is an alphabet, then a Σ-labeled tree is a tree t and a function σ_t from elements of t to Σ. We often use "Σ-tree" for "Σ-labeled tree"; Σ-labeled full binary trees are defined similarly. We often use "tree" or variables t, t', T for "full binary tree" for "labeled tree" or for "full binary labeled tree." It will be clear from the context or unimportant what kind of tree is used. In particular, $t_{\geq u}$ is naturally defined over all such kinds of trees.

Grafting. If t, t_1 are trees and $u \in t$, then the grafting of t_1 at u in t, denoted by $t \circ_u t_1$, is a tree which is obtained from t when $t_{\geq u}$ is replaced by t_1. Formally, this is defined by taking an isomorphic copy t'_1 of t_1 with the domain disjoint from the domain of t and defining a tree order $\leq'$ on $t \setminus t_{\geq u} \cup t'_1$, by $v_1 \leq' v_2$ if v_1 is an ancestor of v_2 in t or in t'_1, or if $v_1 \in t$ and $v_2 \in t'_1$. More generally, if t is a tree, A is an antichain in t and t_1 is a tree, then the grafting of t_1 at A in t is a tree which is obtained from t by replacing every subtree $t_{\geq a}$ by t_1 for $a \in A$. Even more generally, if t is a tree, A is an antichain in t and f assigns a tree t_a to every $a \in A$, then grafting f at A in t is a tree obtained from t, by replacing every subtree $t_{\geq a}$ by t_a for $a \in A$.

Automata. We use standard notations and terminology about Büchi automata on (infinite) full binary Σ-labeled trees and on ω-strings. A Büchi automaton $\mathcal{A}$ has an alphabet Σ, a finite set of states $Q_{\mathcal{A}}$, initial states $Q_I \subseteq Q_{\mathcal{A}}$, a transition relation $\delta_{\mathcal{A}}$, and a set of final states $F \subseteq Q_{\mathcal{A}}$. For a Büchi automaton on ω-string, $\delta_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \Sigma \times Q_{\mathcal{A}}$; for a Büchi tree automaton, $\delta_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \Sigma \times Q_{\mathcal{A}} \times Q_{\mathcal{A}}$.

Given a Büchi automaton $\mathcal{A} = (Q, \Sigma, Q_I, \delta, F)$ and a state $q \in Q$, $\mathcal{A}_q$ is defined as $\mathcal{A}_q = (Q, \Sigma, \{q\}, \delta, F)$, by replacing the set of initial states of $\mathcal{A}$ by $\{q\}$.

The notion of a computation/run, accepting computation of $\mathcal{A}$ on an ω-string/tree is defined as usual. We use letter f for a final state of automata, and we use the letters ϕ, ϕ' for computations. We denote by $ACC(\mathcal{A}, t)$ the set of accepting computations of $\mathcal{A}$ on t. We denote by $L(\mathcal{A}) := \{t \mid ACC(\mathcal{A}, t) \text{ is not empty}\}$ the language of $\mathcal{A}$.

A state q of $\mathcal{A}$ is useful if there is a tree, an accepting computation ϕ on t and a node u such that $\phi(u) = q$. There is a polynomial algorithm that for $\mathcal{A}$ and q checks whether q is useful. There is a polynomial algorithm which for every $\mathcal{A}$ computes an automaton $\mathcal{B}$ with all states useful and $L(\mathcal{A}) = L(\mathcal{B})$. We will always assume that all states are useful.

Degree of Ambiguity. We denote by $|X|$, the cardinality of a set X. $da(\mathcal{A})$ is defined as $\sup\{|ACC(\mathcal{A},t)| \mid t \in L(\mathcal{A})\}$. We say that $\mathcal{A}$ is unambiguous if $da(\mathcal{A}) = 1$, boundedly ambiguous if there is $k \in \mathbb{N}$ such that $da(\mathcal{A}) \leq k$, finitely ambiguous if $|ACC(\mathcal{A},t)|$ is finite for every t, countably ambiguous if $ACC(\mathcal{A},t)$ is countable for every t.

▶ **Example 2.** Consider Büchi tree automata $\{\mathcal{A}_i\}_{i=1}^{3}$ over the unary alphabet.

1. $\mathcal{A}_1$ has a single state q; it is initial and final. $\Delta_1 = (q,q,q)$. $\mathcal{A}_1$ is deterministic.
2. The set of states of $\mathcal{A}_2$ is $Q_2 := \{q, q_1\}$, both are initial and final, and $\Delta_2 := Q \times Q \times Q$. $\mathcal{A}_2$ is uncountably ambiguous.
3. The set of states of $\mathcal{A}_3$ is $Q_3 := \{q, f\}$, f is the final state, q is initial, and $\Delta_3 = \{q\} \times Q \times Q \cup \{(f,f,f)\})$. $\mathcal{A}_3$ is countably ambiguous.
4. Over the unary alphabet there is only one full binary tree; therefore, every finitely ambiguous automata is boundedly ambiguous.

A computation of $\mathcal{A}$ on a Σ-tree t can be considered as a $Q_{\mathcal{A}}$-labeling of t. The grafting of computations $\phi \circ_v \phi'$ is defined as for the corresponding $Q_{\mathcal{A}}$-trees. We often use implicitly the following simple Lemma.

▶ **Lemma 3** (Grafting). *Let $\mathcal{A}$ be an automaton, t, t_1 trees, $v \in t$ and $\phi \in ACC(\mathcal{A},t)$, and $\phi_1 \in ACC(\mathcal{A}_q,t_1)$. If $\phi(v) = q$, then $\phi \circ_v \phi_1$ is an accepting computation of $\mathcal{A}$ on $t \circ_v t_1$.*

A similar lemma holds for general grafting. As an immediate consequence, we obtain the following lemma:

▶ **Lemma 4.** *$da(\mathcal{A}) \geq da(\mathcal{A}_q)$ for every useful state q of $\mathcal{A}$.*

We suspect that the following lemma is folklore. For lack of reference, we provide a proof in the full version of the paper.

▶ **Lemma 5.** *It is computable in polynomial time whether a Büchi tree automaton is unambiguous.*

The next definition and theorem are taken from [8, 10]. They provide a forbidden pattern characterization of degrees of ambiguity of automata on ω-words.

▶ **Definition 6** (Forbidden pattern for ω-word automata). *Let $\mathcal{B}$ be a Büchi automaton on ω-words such that all its states are useful.*

- *$\mathcal{B}$ contains a* forbidden pattern for bounded ambiguity *if there are distinct states p, q such that for a (finite) word u, there are runs of $\mathcal{B}_p$ on u from p to p and from p to q and there is a run of $\mathcal{B}_q$ on u from q to q.*
- *$\mathcal{B}$ contains a* forbidden pattern for countable ambiguity *if there is a final state f and there are two distinct runs of $\mathcal{B}_f$ on the same word u from f to f.*
- *$\mathcal{B}$ contains a* forbidden pattern for finite ambiguity *if it contains the forbidden pattern for countable ambiguity or there is a final state f, and $q \neq f$, and a word u such that there are runs of $\mathcal{B}_q$ on u from q to q and on u from q to f and a run of $\mathcal{B}_f$ on u from f to f.*

▶ **Theorem 7.** *Let $\mathcal{B}$ be a Büchi automaton on ω-words.*

1. *$\mathcal{B}$ has uncountably many accepting runs on some ω-word if and only if $\mathcal{B}$ contains the forbidden pattern for countable ambiguity.*
2. *$\mathcal{B}$ has infinitely many accepting runs on some ω-word if and only if $\mathcal{B}$ contains the forbidden pattern for finite ambiguity.*
3. *$\mathcal{B}$ is not boundedly ambiguous iff it contains the forbidden pattern for bounded ambiguity.*

3 Main Result

In this section we first introduce branch ambiguity and ambiguous transition patterns and then state our main results.

3.1 Branch Ambiguity

▶ **Definition 8** (Projection of a computation on a branch). *Let $\phi \in ACC(\mathcal{A}, t)$ and $\pi := v_0 v_1 \dots$ be a branch of t. $\phi(\pi) := \phi(v_0)\phi(v_1)\dots \in Q_{\mathcal{A}}^{\omega}$ is the projection of ϕ on π. We define $ACC(\mathcal{A}, t, \pi) := \{\phi(\pi) \mid \phi \in ACC(\mathcal{A}, t)\}$.*

▶ **Definition 9** (Branch ambiguity). *$\mathcal{A}$ is at most n branch-ambiguous if $|ACC(\mathcal{A}, t, \pi)| \leq n$ for every t and branch π. $\mathcal{A}$ is bounded branch ambiguous if it is at most n branch ambiguous for some n. $\mathcal{A}$ is finitely (countably) branch ambiguous if $|ACC(\mathcal{A}, t, \pi)|$ is finite (respectively, countable) for every t and π.*

Let $\mathcal{A}$ be a Büchi tree automaton. We define a Büchi ω-automaton $\mathcal{A}_B$ which has the same ambiguity as branch ambiguity of $\mathcal{A}$:

▶ **Definition 10** (Branch automaton). *For a Büchi tree automaton $\mathcal{A} = (Q, \Sigma, Q_I, \delta, F)$, the corresponding branch automaton $\mathcal{A}_B$ is an ω-word automaton $(Q, \Sigma_B, Q_I, \delta_B, F)$, where*

1. *$\Sigma_B := \Sigma \times \Sigma_d \times \Sigma_{cons}$ with*
   - **a.** *$\Sigma_d := \{l, r\}$ directions alphabet (left/right).*
   - **b.** *$\Sigma_{cons} := \{S \subseteq Q \mid \bigcap_{q \in S} L(\mathcal{A}_q) \neq \emptyset\}$ sets of states, which we consider "consistent."*
2. *$(q, a, q') \in \delta_B$ iff $a = (\sigma, l, S)$ and $\exists p \in S : (q, \sigma, (q', p)) \in \delta$ or $a = (\sigma, r, S)$ and $\exists p \in S : (q, \sigma, (p, q')) \in \delta$.*

▶ **Lemma 11.** *The branch ambiguity of a tree automaton $\mathcal{A}$ is bounded (respectively, finite, countable) iff the ambiguity of the corresponding branch ω-automaton $\mathcal{A}_B$ is bounded (respectively, finite, countable).*

▶ **Proposition 12** (Computability of branch ambiguity). *It is computable in polynomial time whether the branch ambiguity of $\mathcal{A}$ is bounded, finite, or countable.*

3.2 Ambiguous Transition Pattern

▶ **Definition 13** (Ambiguous transition pattern). *Let $\mathcal{A} = (Q, \Sigma, Q_I, \delta, F)$ be a Büchi automaton with corresponding branch automaton $\mathcal{A}_B = (Q, \Sigma_B, Q_I, \delta_B, F)$. $\mathcal{A}$ has a* ***q-ambiguous transition pattern*** *if $q \in Q$ and there are $p_1, p_2 \in Q$ and $y_1 \in \Sigma_B^*$, $y_2 \in \Sigma_B^+$ with runs of $\mathcal{A}_B$ from q to p_1 on y_1 and from p_2 to q on y_2 such that at least one of the following holds:*

1. *There are two transitions $(p_1, (a, d, \{q_1\}), p_2), (p_1, (a, d, \{q_2\}), p_2) \in \delta_B$ with $q_1 \neq q_2$ and $L(\mathcal{A}_{q_1}) \cap L(\mathcal{A}_{q_2}) \neq \emptyset$, or*
2. *There is a transition $(p_1, (a, d, \{q_1\}), p_2) \in \delta_B$ with $da(\mathcal{A}_{q_1}) > 1$.*

$\mathcal{A}$ is said to have an ***ambiguous transition pattern*** *if there exists $q \in Q$ such that $\mathcal{A}$ has a q-ambiguous transition pattern.*

Lemma 14 provides sufficient conditions for an ambiguous transition pattern.

▶ **Lemma 14** (q-ambiguous transition pattern)**.** *Let $\mathcal{A}$ be a Büchi tree automaton and $v \perp w$. If one of the following conditions holds, then $\mathcal{A}$ has a q-ambiguous transition pattern.*

1. *There is $\phi \in ACC(\mathcal{A}_q, t)$ such that $q = \phi(v)$ and $\phi(w) = p$, where $\mathcal{A}_p$ is ambiguous.*
2. *There are $\phi, \phi' \in ACC(\mathcal{A}_q, t)$ such that $q = \phi(v)$ and $\forall v'(v' \leq v) \to (\phi(v') = \phi'(v')$, and $\phi(w) \neq \phi'(w)$.*

▶ **Lemma 15.** *If $\mathcal{A}$ has an ambiguous transition pattern then its ambiguity degree is not bounded. If $\mathcal{A}$ has an f-ambiguous transition pattern (for a final state f), then its ambiguity degree is not countable.*

▶ **Lemma 16.** *It is computable in polynomial time whether $\mathcal{A}$ has an ambiguous transition pattern and whether $\mathcal{A}$ has an f-ambiguous transition pattern for a final state f.*

3.3 Characterizations of Degrees of Ambiguity

The next two propositions characterize bounded and finite ambiguity.

▶ **Proposition 17** (Bounded ambiguity)**.** *The following are equivalent:*

1. *Büchi tree automaton $\mathcal{A}$ is not boundedly ambiguous.*
2. *At least one of the following conditions holds:*
   - **a.** *$\mathcal{A}$ is not bounded branch ambiguous.*
   - **b.** *$\mathcal{A}$ has an ambiguous transition pattern.*

When "Büchi tree automaton $\mathcal{A}$" is replaced by "an automaton $\mathcal{A}$ on finite trees" in Proposition 17 we obtain an instance of a theorem proved by Seidl in [12]. A proof of Proposition 17 is a simple variation of the proof in [12]. It will be sketched in the full paper.

▶ **Proposition 18** (Finite ambiguity)**.** *The following are equivalent:*

1. *Büchi tree automaton $\mathcal{A}$ is not finitely ambiguous.*
2. *At least one of the following conditions holds:*
   - **a.** *$\mathcal{A}$ is not finite branch ambiguous.*
   - **b.** *$\mathcal{A}$ has an f-ambiguous transition pattern for a final state f.*

In order to characterize countable ambiguity, we first introduce branching patterns.

▶ **Definition 19** (A branching pattern for $\mathcal{A}$ over (Q, f))**.** *Let $\mathcal{A}$ be a Büchi tree automaton, f a final state of $\mathcal{A}$ and $Q \subseteq Q_{\mathcal{A}} \setminus \{f\}$, where $Q_{\mathcal{A}}$ are the states of $\mathcal{A}$. A branching pattern M for $\mathcal{A}$ over (Q, f) is a function $\tau_M : Q \to Q \times Q$ and a tuple $(q_1, q_2) \in Q \times Q$.*

▶ **Definition 20** (Realizable branching pattern)**.** *Let t be a full binary tree and $u \perp v$ two nodes of t. A branching pattern M for $\mathcal{A}$ over (Q, f) is realized in t at u, v by computations ϕ_1, ϕ_2, $\{\phi_q \mid q \in Q\}$ if the following holds:*

1. *$\phi_1, \phi_2 \in ACC(\mathcal{A}_f, t)$ and $\phi_1(u) = f = \phi_2(v)$, $\phi_1(v) = q_1$ and $\phi_2(u) = q_2$.*
2. *For each $q \in Q$: $\phi_q \in ACC(\mathcal{A}_q, t)$ and $\tau_M(q) = (\phi_q(v), \phi_q(u))$ and ϕ_q visits an accepting state on both paths from the root of t to u and from the root of t to v.*

In Sects. 5 and 7 we prove the next two propositions. Their proof is more complicated than the proofs of Propositions 17 and 18.

▶ **Proposition 21** (Countable ambiguity). *The following are equivalent:*

1. *Büchi tree automaton $\mathcal{A}$ is not countably ambiguous.*
2. *At least one of the following conditions holds:*
   a. *$\mathcal{A}$ is not countable branch ambiguous.*
   b. *$\mathcal{A}$ has an f-ambiguous transition pattern for a final state f.*
   c. *A branching pattern for $\mathcal{A}$ is realizable.*

▶ **Proposition 22.** *It is computable in polynomial time whether there is a realizable branching pattern for a Büchi tree automaton $\mathcal{A}$.*

▶ **Theorem 23** (Main). *It is computable in polynomial time whether a Büchi tree automaton is unambiguous, bounded ambiguous, finitely ambiguous, or countably ambiguous.*

Proof. For unambiguity – by Lemma 5. For bounded ambiguity by Proposition 17, Lemma 16 and Proposition 12. For finite ambiguity by Proposition 18, Lemma 16 and Proposition 12. For countable ambiguity by Propositions 21, 12, 22 and Lemma 16. ◀

Road map of the proofs. Sect. 4 and Sect. 5 deal with structural characterizations of finite and countable ambiguity and prove Propositions 18 and 21. Sects 6 and 7 deal with computability of degrees of ambiguity. Proposition 12 and Lemma 16 are proved in Sect. 6, and Proposition 22 is proved in Sect. 7. The proofs of Lemmas 11, 14 and 15 are given in the full version of the paper.

4 Finite Ambiguity

In this section we prove Proposition 18 - a structural characterization of finite ambiguity. (2) ⇒ (1) follows from Lemma 11 and Lemma 15. Below we prove the (1) ⇒ (2) direction.

Let t be a tree such that $ACC(\mathcal{A}, t)$ is not finite. We define a branch $\pi := v_0 \dots v_i \dots$ in t and an ω-sequence of states $q_0 \dots q_i \dots$ such that for every i:

1. From q_i there are infinitely many accepting computations of $\mathcal{A}_{q_i}$ on the subtree $t_{\geq v_i}$.
2. There is an accepting computation ϕ_i on t such that $\phi_i(v_j) = q_j$ for every $j \leq i$.

Define v_0 as the root of t and q_0 as an initial state from which there are infinitely many accepting computations.

Assume that v_i and q_i were defined. Since there are infinitely many accepting computations from state q_i on the subtree $t_{\geq v_i}$, infinitely many of them take the same first transition from q_i to $\langle q_l, q_r \rangle$ and either there are infinitely many accepting computations from state q_l on the subtree rooted at the left child of v_i, or from state q_r on the subtree rooted at the right child of v_i. Define v_{i+1} and q_{i+1} according to these cases.

If $|ACC(\mathcal{A}, t, \pi)|$ is infinite, then by the definition of branch ambiguity we have that $\mathcal{A}$ is not finite branch ambiguous, and 2(a) holds. Otherwise, there exist $\phi_1, \dots, \phi_k \in ACC(\mathcal{A}, t)$ such that $ACC(\mathcal{A}, t, \pi) = \{\phi_i(\pi) \mid 1 \leq i \leq k\}$. Choose n such that for all $1 \leq i < j \leq k : \phi_i(v_0 \dots v_n) \neq \phi_j(v_0 \dots v_n)$. One of these computations, say ϕ_1, holds that $\forall i \leq n : \phi_1(v_i) = q_i$. Hence, $\forall i : \phi_1(v_i) = q_i$.

Let f be an accepting state which occurs infinitely often in $\phi_1(\pi)$. Choose $N > n$ such that $\phi_1(v_N) = q_N = f$. By selection of q_N, there are infinitely many accepting computations of $\mathcal{A}_f$ on $t_{\geq v_N}$. Take two different accepting computations $\phi', \phi'' \in ACC(\mathcal{A}_f, t_{\geq v_N})$. By selection of ϕ_1, $\forall i \geq N : \phi_1(v_i) = \phi'(v_i) = \phi''(v_i) = q_i$. Therefore, ϕ' and ϕ'' differ at some node $w \notin \pi$, and there exist $i > N$ such that $\phi_1(v_i) = f = \phi'(v_i) = \phi''(v_i)$ and $v_i \perp w$. Applying Lemma 14(2) on ϕ', ϕ'' and $v_i \perp w$, we obtain that $\mathcal{A}_f$ has an f-ambiguous transition pattern, and 2(b) holds.

5 Countable Ambiguity

In this section we prove Proposition 21 – a structural characterization of countable ambiguity.

5.1 Direction (2) ⇒ (1) of Proposition 21

2(a)⇒(1) follows by definition of branch ambiguity, and 2(b)⇒(1) follows by Lemma 15. Below 2(c)⇒(1) is proved.

▶ **Definition 24** (Corresponding automaton $\mathcal{A}_M$ for pattern M). *Let M be a branching pattern for $\mathcal{A}$ over (Q, f) (see Definition 19). We define a Büchi tree automaton $\mathcal{A}_M$ over the unary alphabet with the set of states $Q \cup \{f\}$; all states are final, the initial state is f, and the transition relation is $\Delta_M := \{(q, q', q'') \mid q \in Q \text{ and } (q', q'') = \tau_M(q)\} \cup \{(f, q_1, f),\ (f, f, q_2)\}$.*

The following simple lemma states the properties of accepting computations of $\mathcal{A}_M$. It will be useful in showing that if a branching pattern for $\mathcal{A}$ is realized, then $\mathcal{A}$ is not countably ambiguous.

▶ **Lemma 25** (Accepting computations of $\mathcal{A}_M$).

1. *Let ϕ be an accepting computation of $\mathcal{A}_M$. Then the set of nodes $\{v \mid \phi(v) = f\}$ is a branch.*
2. *For every branch π there is an accepting computation ϕ of $\mathcal{A}_M$ such that $\forall v \in \pi(\phi(v) = f)$.*
3. *The set of accepting computations of $\mathcal{A}_M$ is uncountable.*

▶ **Lemma 26.** *Let $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, Q_I, \delta, F)$ be a Büchi tree automaton such that a branching pattern for $\mathcal{A}$ is realizable. Then $\mathcal{A}$ is not countably ambiguous.*

Proof. Assume that a branching pattern M over (Q, f) is realized in t at $u \perp v$ by ϕ_1, ϕ_2, $\{\phi_q \mid q \in Q\}$. We construct a sequence of trees: $t_1 := t$, and $\forall i \geq 1 : t_{i+1} := t_i \circ_{A_i} t$, where $A_i = \{u, v\}^i$. We graft t at every node in A_i of t_i. This operation is well defined as A_i is an antichain ($\forall a_1 \neq a_2 \in A_i : a_1 \perp a_2$, since $u \perp v$).

For each $y \in \{l, r\}^*$ we define $k_y := max\{i \mid y \in \{u, v\}^i \cdot z, z \in \{l, r\}^*\}$. Notice that by the construction, if $\sigma_{t_{1+k_y}}(y) = a$ then $\forall i > k_y : \sigma_{t_i}(y) = a$. Define t^ω as $\sigma_{t^\omega}(y) := t_{1+k_y}(y)$.

We now proceed to show that the set of accepting computations of $\mathcal{A}_f$ on t^ω is not countable, by defining an injective map from the set of accepting computations of $\mathcal{A}_M$ (on the tree over the unary alphabet) to the set of accepting computations of $\mathcal{A}_f$ on t^ω.

▶ **Notations 27.** *Let h be a homomorphism $h : \{l, r\}^* \to \{l, r\}^*$, where $h(l) = v$ and $h(r) = u$. Since $u \perp v$, it follows that h is a bijection from $\{l, r\}^*$ onto $\{u, v\}^*$.*

For each accepting computation ϕ of $\mathcal{A}_M$ we assign an accepting computation $\widehat{\phi}$ of $\mathcal{A}_f$ on t^ω. If $w \in \{u, v\}^*$ then $\widehat{\phi}(w) := \phi(h^{-1}(w))$ (hence, the map is injective). Otherwise, let $w = y \cdot z$ where $y \in \{u, v\}^{k_w}$ and $z \in \{l, r\}^+$. If $\phi(h^{-1}(y)) = q \neq f$ then $\widehat{\phi}(w) := \phi_q(z)$. Else, if $\phi(h^{-1}(y \cdot u)) = f$ then $\widehat{\phi}(w) := \phi_1(z)$; otherwise, $\widehat{\phi}(w) := \phi_2(z)$ (recall that ϕ_1, ϕ_2, ϕ_q are computations on t that realize M).

It is routine to verify that $\widehat{\phi}$ is an accepting computation of $\mathcal{A}_f$ on t^ω. By Lemma 25, $\mathcal{A}_M$ has uncountably many accepting computations and we defined an injective map from these computations to accepting computations of $\mathcal{A}_f$. Hence, $\mathcal{A}_f$ is not countably ambiguous. Therefore, by Lemma 4, $\mathcal{A}$ is not countably ambiguous. ◀

5.2 Direction (1) ⇒ (2) of Proposition 21

▶ **Definition 28** (q-path and q-computation)**.** *Given a Büchi tree automaton $\mathcal{A} = (Q, \Sigma, Q_I, \delta, F)$, a state $q \in Q$ and a tree $t \in L(\mathcal{A})$, we define the following:*

- *A **q-path** (of an accepting computation ϕ) is an ω-path $\pi := v_0 \dots v_i \dots$ of t such that v_0 is the root, $\phi(v_0) = q$ and there exist infinitely many nodes v_i such that $\phi(v_i) = q$.*
- *A **q-computation** is an accepting computation ϕ such that ϕ has a q-path in t.*

The next lemma reduces the question whether the cardinality of accepting computations is uncountable to the question whether the cardinality of f-computations is uncountable.

▶ **Lemma 29.** *A Büchi tree automaton $\mathcal{A} = (Q, \Sigma, Q_I, \delta, F)$ has uncountably many accepting computations on t iff there is a state $f \in F$, a node $u \in t$ and an accepting computation $\phi_0 \in ACC(\mathcal{A}, t)$ such that $\phi_0(u) = f$ and $\mathcal{A}_f$ has uncountably many f-computations on $t_{\geq u}$.*

Proof. ⇐ direction is trivial.

⇒: Assume that the set $\Phi := ACC(\mathcal{A}, t)$ of accepting computations of $\mathcal{A}$ on t is uncountable. For each computation $\phi \in \Phi$ define a tree t'_ϕ by pruning the tree t as follows: for every node $v \in t$, if $\phi(v) \in F$ and ϕ has an $\phi(v)$-path on $t_{\geq v}$, prune the descendants of v. Hence, t'_ϕ is a subtree of t over the set of nodes $V_\phi := \{v \mid \forall u < v : \phi(u) \in F \rightarrow \phi$ has no $\phi(u)$-paths on $t_{\geq u}\}$. If u is a leaf of t'_ϕ, then $\phi(u) \in F$ and ϕ has an $\phi(u)$-path on $t_{\geq u}$.

Observe that t'_ϕ is finite. Otherwise, by the König Lemma, it would have an infinite branch $\pi = v_0 \dots v_i \dots$ such that $\phi(v_0) \dots \phi(v_i) \dots$ has finitely many occurrences of states from F which contradicts that ϕ is an accepting run on t.

Therefore, to each computation $\phi \in \Phi$ corresponds a finite tree t'_ϕ. The set of all possible finite trees is countable, and since there are uncountably many computations in Φ, we conclude that there is a finite tree t_0 and an uncountable set $\Phi_{t_0} \subseteq \Phi$ such that $\forall \phi \in \Phi_{t_0} : t_0 = t'_\phi$. Since there are finitely many assignments of states to the nodes of t_0, we conclude that there is a computation ϕ_0 and an uncountable set $\Phi' \subseteq \Phi_{t_0}$ such that $\forall v \in t_0 \forall \phi \in \Phi' : \phi(v) = \phi_0(v)$. For each leaf $u \in t_0$ define Φ_u as the set of restrictions Φ' on $t_{\geq u}$. Notice that the cardinality of Φ' is bounded by the product of the cardinalities of Φ_u. Hence, there is u such that Φ_u is uncountable. Each computation $\phi \in \Phi_u$ has originated from a computation with an $\phi_0(u)$-path on $t_{\geq u}$, and therefore Φ_u is the set of f-computations of $\mathcal{A}_f$ on $t_{\geq u}$ for $f = \phi_0(u)$. ◀

We are going to prove that if $\mathcal{A}$ is not countably ambiguous and has at most countable branch ambiguity and no f-ambiguous transition pattern, then it has a branching pattern. The main technical lemma uses the following definition.

▶ **Definition 30.** *Let T be a subset of nodes of a tree t. We consider T as a substructure of t with the ancestor relation. In particular, u is a **T-leaf** if $u < v$ for no $v \in T$; u is a **T-successor** of v if $u, v \in T$, $u > v$ and no T-node is between u and v; T is a **full binary subset-tree** of t, if T has a minimal node and every node of T has two T-successors.*

▶ **Lemma 31** (Main)**.** *Assume f is a final state and there are uncountably many f-computations of $\mathcal{A}$ on t, and conditions 2(a) and 2(b) of Proposition 21 do not hold. Then, there is a full binary subset-tree X of t such that for every $u \in X$ there is an f-computation ϕ_u on t such that if $v \in X$ and $v \leq u$ then $\phi_u(v) = f$.*

The lemma is proved in the full paper, where X is constructed level by level. However, in order to carry out such a construction we need a much stronger inductive assertion. In particular, our construction implies that for every $u \in X$, $\mathcal{A}_f$ has uncountably many f-computations on $t_{\geq u}$. The next lemma is easily derived from the König Lemma.

▶ **Lemma 32.** *If T is a full binary subset-tree of t, then there is a full binary subset-tree $T' \subseteq T$ such that if v_1, v_2 are the T'-successors of u, and $\mathcal{A}_q$ accepts $t_{\geq u}$, then $\mathcal{A}_q$ has an accepting computation on $t_{\geq u}$ which passes through F on the paths of $t_{\geq u}$ from u to v_1 and from u to v_2.*

▶ **Lemma 33.** *Let $(T, \leq)$ be the full binary tree and lab be a labeling of its nodes by a finite alphabet. Then, there are $v_1, v_2 > u$ such that $v_1 \perp v_2$ and $lab(v_1) = lab(u) = lab(v_2)$.*

Proof. Choose a node u such that the cardinality of $\Sigma_{\geq u} := \{lab(w) \mid u \leq w\}$ is minimal. Then for every $w' \geq u$ and every $\sigma \in \Sigma_{\geq w}$ there is $v' \geq w'$ with $lab(v') = \sigma$. ◀

The next Lemma, together with Lemma 29, shows that if a tree automaton is uncountably ambiguous and 2(a) and 2(b) of Proposition 21 do not hold, then 2(c) holds. This implies the (1) ⇒ (2) direction of Proposition 21.

▶ **Lemma 34.** *Let $\mathcal{A}$ be a Büchi tree automaton and f be a final state of $\mathcal{A}$. Assume that there are uncountably many f-computations of $\mathcal{A}_f$ on t and conditions 2(a) and 2(b) of Proposition 21 do not hold. Then, there exist three nodes $u, v_1, v_2 \in t$ such that a branching pattern for $\mathcal{A}_f$ is realized at v_1, v_2 in $t_{\geq u}$.*

Proof. Let X be the full binary subset-tree of t, guaranteed by Lemma 31. By applying Lemma 32 on X, we obtain a full binary subset-tree $T \subseteq X$. Define a labeling of T by $lab(v) = \{\phi(v) | \phi \in ACC(\mathcal{A}_f, t)\}$ for each $v \in T$. This is a labeling by a finite alphabet. Therefore, by Lemma 33, we have nodes $v_1, v_2 > u$ such that $v_1 \perp v_2$ and $lab(u) = lab(v_1) = lab(v_2) = Q'$. We are going to define computations that realize a branching pattern over $(Q' \setminus \{f\}, f)$ at v_1, v_2 in $t_{\geq u}$.

For $i = 1, 2$, set ϕ_i to be the restriction of ϕ_{v_i} to $t_{\geq u}$, where ϕ_{v_i} is as in Main Lemma. This gives immediately that $\phi_i \in ACC(\mathcal{A}_f, t_{\geq u})$ and $\phi_1(v_1) = \phi_2(v_2) = f$. Since A_f is ambiguous, by Lemma 14(1) and the assumption that $\mathcal{A}$ has no f-ambiguous transition pattern, we obtain $\phi_1(v_2) \neq f \neq \phi_2(v_1)$.

By Lemma 32, for each $q \in Q' \setminus \{f\}$ there is $\phi_q \in ACC(\mathcal{A}_q, t_{\geq u})$ which visits F on the paths (in $t_{\geq u}$) from u to the children of u in T. Hence, it visits F on the paths from u to v_1 and from u to v_2. Next, observe that $\phi_q(v_1), \phi_q(v_2) \in Q'$ by the definition of labeling. We are going to show that $\phi_q(v_1) \neq f$ and $\phi_q(v_2) \neq f$. This will show that ϕ_1, ϕ_2 and ϕ_q for $q \in Q' \setminus \{f\}$ realize a branching pattern, and thus finish the proof.

Aiming for a contradiction, assume $\phi_q(v_1) = f$. There is $\phi' \in ACC(A_f, t)$ such that $\phi'(u) = q$. Let ϕ'_q be a grafting of ϕ_q on ϕ' at u. It reaches v_1 in state f. Consider the branch automaton computation from the root of t to v_1 which correspond to ϕ'_q and ϕ_{v_1}. These are different computations (since they differ at u) from f to f. Hence, $\mathcal{A}_B$ contains the forbidden pattern for countable ambiguity (see Def. 6), and by Theorem 7 we have that $\mathcal{A}_B$ is not countably ambiguous. Therefore, $\mathcal{A}$ is not countable branch ambiguous - contradiction to the assumptions of the lemma. The proof of $\phi_q(v_2) \neq f$ is similar. ◀

6 Computability of branch ambiguity and the ambiguous transition pattern

Here we describe algorithms to test the degree of ambiguity of branch automata and to test if an automaton has an ambiguous transition pattern. The following Lemma easily follows from Definition 10 of the branch automaton.

▶ **Lemma 35.** *Let $\mathcal{A}_B$ be the branch automaton of $\mathcal{A}$. Assume that $r_i \in Q^{l+1}$ for $i = 1, \dots, k$ are runs of $\mathcal{A}_B$ on $u = (\sigma_1, d_1, S_1) \dots (\sigma_l, d_l, S_l) \in \Sigma_B^*$. Then for $i = 1, \dots, l$ there are $S'_i \subseteq S_i$ such that $|S'_i| \leq k$ and r_i for $i = 1, \dots, k$ are runs of $\mathcal{A}_B$ on $u = (\sigma_1, d_1, S'_1) \dots (\sigma_l, d_l, S'_l)$.*

A letter $(\sigma, d, S) \in \Sigma_B$ is called a k-state letters if S has at most k states. If $\mathcal{A}$ has n states, then the alphabet Σ_B of the branch automaton $\mathcal{A}_B$ might be of size $2|\Sigma| \times 2^n$, yet the number of k-state letter is bounded by $2|\Sigma| \times \sum_{i=1}^{k} \binom{n}{i} \leq 2|\Sigma| n^k$. To test whether a k-state letter (σ, d, S) is in Σ_B, we can check whether the intersection of the tree languages $L(\mathcal{A}_q)$ for $q \in S$ is non-empty. This can be done in $O(n^k)$ time (checking non-emptiness of the intersection Büchi language). We denote by $\mathcal{A}_B^{(k)}$ the restriction of the branch automaton $\mathcal{A}_B$ to the k state letters. It is computable in $O(|\mathcal{A}|^k)$ time from $\mathcal{A}$.

Now, we are ready to prove Lemma 16 and Proposition 12.

Proof of Lemma 16. For each p_1 and p_2, items 1 and 2 of Definition 13 can be tested in polynomial time. There is a q-ambiguous pattern, if there is a run of $\mathcal{A}_B^{(1)}$ from q to p_1 and from p_2 to q for a pair p_1 and p_2 which passed the test. This is reduced to the reachability problem. ◀

Proof Sketch of Proposition 12. The degree of ambiguity of ω-word Büchi automata is characterized by the forbidden patterns in Theorem 7. Each of these patterns involves conditions on at most three runs on the same word and can be tested for an automaton $\mathcal{B}$ in polynomial time. Hence, by Lemma 35, $\mathcal{A}_B$ has these patterns iff $\mathcal{A}_B^{(3)}$ has them, and can be tested in time $p(|\mathcal{A}_B^{(3)}|)$ for a polynomial p. Since $\mathcal{A}_B^{(3)}$ is computable in polynomial time from $\mathcal{A}$, we obtain a polynomial time algorithm. ◀

7 Computability of a branching pattern

Here we prove Proposition 22. In Sect. 7.1 we show that if $\mathcal{A}$ has a branching pattern, then it has a branching pattern over (Q, f), where Q has at most two states. Sect. 7.2 presents a polynomial time algorithm to verify if $\mathcal{A}$ has a branching pattern with at most two states.

7.1 Reduction to small branching patterns

In Sect. 5.1 we assigned to each branching pattern M a tree automaton $\mathcal{A}_M$ over the unary alphabet. This automaton is almost deterministic, in the sense that from every state $q \neq f$ it has a unique transition and it does not enter f. Hence, $\mathcal{A}_M$ has a unique accepting computation from every $q \neq f$. From f it has two transitions. A transition function defined next will help to describe properties of accepting computations of $\mathcal{A}_M$.

▶ **Definition 36** (Transition function of branching pattern)**.** *Let M be a branching pattern for $\mathcal{A}$ over (Q, f) with $\tau_M : Q \to Q \times Q$ and a tuple $(q_1, q_2) \in Q \times Q$. Its transition function $\delta_M : (\{f\} \cup Q) \times \{l, r\} \to Q$ is defined as follows:*

$$\delta_M(f, d) := \begin{cases} q_1 & \text{if } d = l \\ q_2 & \text{if } d = r \end{cases} \quad ;\text{For } p \neq f, \text{with } \tau_M(p) = (q', q'') : \quad \delta_M(p, d) := \begin{cases} q' & \text{if } d = l \\ q'' & \text{if } d = r \end{cases}$$

▶ **Lemma 37.**

1. *Let $q \neq f$ and ϕ^q be a (unique) accepting computation of $\mathcal{A}_M$ (on the tree over unary alphabet) from q. Then $\phi^q(w) = \delta_M(q, w)$ for every $w \in \{l, r\}^*$.*
2. *Let $s := d_1 \dots d_k \in \{l, r\}^+$, and let ϕ_s be an accepting computation of $\mathcal{A}_M$ from f such that $\phi_s(d_1 \dots d_i) = f$ for every $i \leq k$. Then for every $w \in \{l, r\}^*$: (a) if $d_i = l$ then $\phi_s(d_1 \dots d_{i-1} r w) = \delta_M(f, l w)$ and (b) if $d_i = r$ then $\phi_s(d_1 \dots d_{i-1} l w) = \delta_M(f, r w)$.*

▶ **Lemma 38.** *Assume a branching pattern M for $\mathcal{A}$ over (Q, f) is realized. Let $l_M(q) := \delta_M(q, l)$ and $r_M(q) := \delta_M(q, r)$ for all $q \in Q$. Then:*

1. *If l_M maps Q to $Q_0 \subsetneq Q$, then a branching pattern for $\mathcal{A}$ over (Q_0, f) is realized. Dually, if r_M maps Q to $Q_1 \subsetneq Q$ then a branching pattern for $\mathcal{A}$ over (Q_1, f) is realized.*
2. *If l_M and r_M are bijections, then there is Q' such that $|Q'| \leq 2$ and a branching pattern for $\mathcal{A}$ over (Q', f) is realized.*
3. *A branching pattern for $\mathcal{A}$ over (Q', f) is realized with $|Q'| \leq 2$.*

Proof. We will assume the branching pattern M for $\mathcal{A}$ over (Q, f) is realized in a tree t at nodes u, v by computations ϕ_1, ϕ_2, $\{\phi_q \mid q \in Q\}$.

(1) Assume l_M maps Q to $Q_0 \subsetneq Q$. Let $t' := (t \circ_u t) \circ_v t$. Define the following computations on t': $\phi'_1 := (\phi_1 \circ_u \phi_2) \circ_v \phi_{q_1}$, $\phi'_2 := (\phi_2 \circ_u \phi_{q_2}) \circ_v \phi_2$, and for each $q \in Q_0$ with $f_M(q) = (p_1, p_2)$ we set $\phi'_q := (\phi_q \circ_u \phi_{p_2}) \circ_v \phi_{p_1}$. Let $u' := uv$ and $v' = vv$ be two nodes of t'. By Lemma 3 we have that $\phi'_1, \phi'_2 \in ACC(\mathcal{A}_f, t)$ and $\forall q \in Q_0 : \phi'_q \in ACC(A_q, t')$. Notice that $\phi'_1(u') = \phi'_1(uv) = \phi_2(v) = f$, $\phi'_2(v') = \phi'_2(vv) = \phi_2(v) = f$, and from the construction it follows that $\phi'_1(v'), \phi'_2(u'), \phi'_q(u'), \phi'_q(v') \in \{\phi_q(v) \mid q \in Q\} = \{\delta_M(q, l) \mid q \in Q\} \subseteq Q_0$. Since ϕ_q visits F on both paths from the root to u and from the root to v, so does ϕ'_q on the path from the root to $u' = uv$ and from the root to $v' = vv$. It follows that a branching pattern for $\mathcal{A}$ over (Q_0, f) is realized in t' at u', v' by computations ϕ'_1, ϕ'_2 and $\{\phi'_q \mid q \in Q_0\}$. The proof of the dual case is symmetric.

(2) The set of bijections on a finite set is a finite group under the composition and the identity map is its identity element. If k is the cardinality of a finite group, then c^k is equal to the identity for every element c. Let $k > 0$ be such that both l^k_M and r^k_M are the identity map.

Define $t^u_1 := t$, $t^v_1 := t$ and $\forall i > 1$ let $t^u_{i+1} := t \circ_u t^u_i$ and $t^v_{i+1} := t \circ_v t^v_i$. Finally, construct a tree $t' := (t \circ_u t^u_{k-1}) \circ_v t^v_{k-1}$.

Let $p_1 := \delta_M(f, l^k)$ and $p_2 := \delta_M(f, r^k)$. We will show that a branching pattern for $\mathcal{A}$ over $(\{p_1, p_2\}, f)$ is realized in t' at u^k, v^k.

The following are obtained using Lemma 3 and definition of δ_M:

i $\alpha_i := \underbrace{\phi_1 \circ_u (\phi_1 \circ_u (\cdots \circ_u \phi_1) \dots)}_{i \text{ times}}$ is an accepting computation of $\mathcal{A}_f$ on t^u_i. It assigns f to node u^i.

ii $\beta_i := \underbrace{\phi_2 \circ_v (\phi_2 \circ_v (\cdots \circ_v \phi_2) \dots)}_{i \text{ times}}$ is an accepting computation of $\mathcal{A}_f$ on t^v_i. It assigns f to node v^i.

iii Let $q_0 \in Q$ and $q_i := \delta_M(q_0, r^i)$. Then $\phi^{r^i}_{q_0} := \phi_{q_0} \circ_u (\phi_{q_1} \circ_u (\cdots \circ_u \phi_{q_{i-1}}) \dots)$ is an accepting computation of $\mathcal{A}_{q_0}$ on t^u_i, which holds $\phi^{r^i}_{q_0}(u^j) = q_j$ for $j \leq i$.

iv Let $q'_0 \in Q$ and $q'_i := \delta_M(q'_0, l^i)$. Then $\phi^{l^i}_{q'_0} := \phi_{q'_0} \circ_v (\phi_{q'_1} \circ_v (\cdots \circ_v \phi_{q'_{i-1}}) \dots)$ is an accepting computation of $\mathcal{A}_{q'_0}$ on t^v_i, which holds $\phi^{l^i}_{q'_0}(v^j) = q'_j$ for $j \leq i$.

Let $q' := \phi_1(v)$ and $q'' := \phi_2(u)$. From **i** and **iv**, it follows that $\phi'_1 := (\phi_1 \circ_u \alpha_{k-1}) \circ_v \phi^{l^{k-1}}_{q'}$ is an accepting computation of $\mathcal{A}_f$ on t', such that $\phi'_1(u^k) = f$, $\phi'_1(v^k) = \delta_M(f, l^k)$ and ϕ'_1 visits F on the path from the root to v^k (as it coincides with ϕ_1 on the path from the root to v, which visits F).

Using similar arguments from **ii** and **iii**, we obtain that $\phi'_2 := (\phi_2 \circ_u \phi^{r^{k-1}}_{q''}) \circ_v \beta_{k-1}$ is an accepting computation of $\mathcal{A}_f$ on t', such that $\phi'_2(v^k) = f$, $\phi'_2(u^k) = \delta_M(f, r^k)$ and ϕ'_2 visits F on the path from the root to u^k.

In addition, from **iii** and **iv** is follows that for all $p \in Q$ with $\tau_M(p) = (p', p'')$, the computation $\phi'_p := (\phi_p \circ_u \phi^{r^{k-1}}_{p''}) \circ_v \phi^{l^{k-1}}_{p'}$ is an accepting computation of $\mathcal{A}_p$ on t', such that $\phi'_p(u^k) = \delta_M(p, r^k)$ and $\phi'_p(v^k) = \delta_M(p, l^k)$. By selection of k we have $\delta_M(p, l^k) = \delta_M(p, r^k) = p$ and therefore we obtain that $\phi'_p(u^k) = p = \phi'_p(v^k)$.

Take $p_1 := \delta_M(f, l^k) = \phi'_1(v^k)$ and $p_2 := \delta_M(f, r^k) = \phi'_2(u^k)$. We have $\phi'_{p_1}(u^k) = \phi'_{p_1}(v^k) = p_1$ and $\phi'_{p_2}(u^k) = \phi'_{p_2}(v^k) = p_2$, and therefore we obtain that a branching pattern for $\mathcal{A}$ over $(\{p_1, p_2\}, f)$ is realized in t' at u^k, v^k by computations ϕ'_1, ϕ'_2, ϕ'_{p_1} and ϕ'_{p_2}, as requested.

(3) Let M over (Q, f) be a realizable branching pattern for $\mathcal{A}$ such that the cardinality of Q is minimal. If either l_M or r_M is not a bijection, then by item 1, there is a realizable pattern over (Q_0, f), where $|Q_0| < |Q|$. Hence, both l_M and r_M are bijections. Therefore, by item 2 and minimality of $|Q|$, we obtain $|Q| \leq 2$. ◀

7.2 Small branching patterns are in P

For every t over Σ and $u_1, u_2 \in t$, define $t' := t(u_1, u_2)$ over $\Sigma' := \Sigma \times \Sigma_{u_1} \times \Sigma_{u_2}$ with $\Sigma_{u_i} := \{0, 1\}$, such that the projection of t' on Σ is t and the projection of t' on Σ_{u_i} is a tree t_{u_i} with $\sigma_{t_{u_i}}(w) = 1$ iff $w = u_i$ for $i = 1, 2$.

It is easy to construct Büchi automata over Σ' with the following properties in $O(|\mathcal{A}|)$ time.

- An automaton $\mathcal{A}_{nodes}$ which accepts t' iff $t' = t(u_1, u_2)$ and $u_1 \perp u_2$.
- An automaton $\mathcal{A}_{q,q_1,q_2}$ which accepts t' iff $t' = t(u_1, u_2)$ and there exists a computation $\phi \in ACC(\mathcal{A}_q, t)$ with $\phi(u_1) = q_1$, $\phi(u_2) = q_2$ and ϕ visits an accepting state on both paths from the root to u_1 and from the root to u_2.
- An automaton $\mathcal{A}^L_{f,q}$ which accepts t' iff $t' = t(u_1, u_2)$ and there exists a computation $\phi \in ACC(\mathcal{A}_f, t)$ such that $\phi(u_1) = f$ and $\phi(u_2) = q$.
- An automaton $\mathcal{A}^R_{f,q}$ which accepts t' iff $t' = t(u_1, u_2)$ and there exists a computation $\phi \in ACC(\mathcal{A}_f, t)$ such that $\phi(u_1) = q$ and $\phi(u_2) = f$.

By Lemma 38, $\mathcal{A}$ has a realizable branching pattern iff there exists a realizable branching pattern over (Q, f), $\tau_M : Q \to Q$, $(q_1, q_2) \in Q \times Q$ with $|Q| \leq 2$. For each such branching pattern we define:

$$L_M := L(\mathcal{A}_{nodes}) \cap \bigcap_{(p,p_1,p_2)|p\in Q,\tau_M(p)=(p_1,p_2)} L(\mathcal{A}_{p,p_1,p_2}) \cap L(\mathcal{A}^L_{f,q_1}) \cap L(\mathcal{A}^R_{f,q_2})$$

By construction of the automata we have that the branching pattern M is realizable iff $L_M \neq \emptyset$. This could be verified in polynomial time in $|Q_{\mathcal{A}}|$, as this is an intersection of at most five Büchi tree languages. Since the number of such patterns is polynomial in $|Q_{\mathcal{A}}|$ we obtain a polynomial time algorithm.

8 Conclusion and Further Results

We proved that the degree of ambiguity of Büchi automata on infinite trees is in PTIME. The Büchi acceptance conditions on trees are less expressive than parity, Rabin, Street and Muller conditions. Unfortunately, we have

▶ **Proposition 39.**

- *The problems of deciding whether a parity tree automaton is not unambiguous/boundedly ambiguous/finitely ambiguous are co-NP complete*
- *The problem of deciding weather a parity tree automaton is not countably ambiguous is co-NP hard.*

It is still unknown if the problem of deciding weather a parity tree automaton is not countably ambiguous is in co-NP, although we believe it is indeed the case.

The degree of ambiguity of a regular language is defined in a natural way. E.g., a language is k-ambiguous if it is accepted by a k-ambiguous automaton and no $k-1$-ambiguous automaton accepts it. Over finite words and finite trees every regular language is accepted by a deterministic automaton. Over ω-words every regular language is accepted by an unambiguous automaton [1]. Over infinite tree there are ambiguous languages [3]. We can show that over infinite trees there is a hierarchy of degrees of ambiguity [11]:

▶ **Proposition 40.** *There are k-ambiguous languages for every $k \in \mathbb{N}$. There are finitely, countably and uncountable ambiguous languages.*

We plan to investigate whether the degree of ambiguity of infinite tree language is decidable.

References

1 André Arnold. Rational omega-Languages are Non-Ambiguous. *Theor. Comput. Sci.*, 26:221–223, September 1983. doi:10.1016/0304-3975(83)90086-5.

2 Vince Bárány, Łukasz Kaiser, and Alex Rabinovich. Expressing cardinality quantifiers in monadic second-order logic over trees. *Fundamenta Informaticae*, 100(1-4):1–17, 2010.

3 Arnaud Carayol, Christof Löding, Damian Niwinski, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Open Mathematics*, 8(4):662–682, 2010.

4 Thomas Colcombet. Unambiguity in automata theory. In *International Workshop on Descriptional Complexity of Formal Systems*, pages 3–18. Springer, 2015.

5 Yo-Sub Han, Arto Salomaa, and Kai Salomaa. Ambiguity, nondeterminism and state complexity of finite automata. *Acta Cybernetica*, 23(1):141–157, 2017.

6 Ernst Leiss. Succinct representation of regular languages by Boolean automata. *Theoretical computer science*, 13(3):323–330, 1981.

7 Hing Leung. Descriptional complexity of NFA of different ambiguity. *International Journal of Foundations of Computer Science*, 16(05):975–984, 2005.

8 Christof Löding and Anton Pirogov. On Finitely Ambiguous Büchi Automata. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, pages 503–515, 2018. doi:10.1007/978-3-319-98654-8_41.

9 Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games*, volume 141. Academic Press, 2004.

10 Alexander Rabinovich. Complementation of Finitely Ambiguous Büchi Automata. In *International Conference on Developments in Language Theory*, pages 541–552. Springer, 2018.

11 Alexander Rabinovich and Doron Tiferet. Degree of Ambiguity for Tree Automata and Tree Languages, forthcoming.

12 Helmut Seidl. On the finite degree of ambiguity of finite tree automata. *Acta Informatica*, 26(6):527–542, 1989.

13 Wolfgang Thomas. Automata on infinite objects. In *Formal Models and Semantics*, pages 133–191. Elsevier, 1990.

14 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.

Regular Separability and Intersection Emptiness Are Independent Problems

Ramanathan S. Thinniyam
Max Planck Institute for Software Systems (MPI-SWS), Germany
thinniyam@mpi-sws.org

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Germany
georg@mpi-sws.org

Abstract

The problem of *regular separability* asks, given two languages K and L, whether there exists a regular language S that includes K and is disjoint from L. This problem becomes interesting when the input languages K and L are drawn from language classes beyond the regular languages. For such classes, a mild and useful assumption is that they are full trios, i.e. closed under rational transductions.

All the results on regular separability for full trios obtained so far exhibited a noteworthy correspondence with the intersection emptiness problem: In each case, regular separability is decidable if and only if intersection emptiness is decidable. This raises the question whether for full trios, regular separability can be reduced to intersection emptiness or vice-versa.

We present counterexamples showing that neither of the two problems can be reduced to the other. More specifically, we describe full trios $\mathcal{C}_1$, $\mathcal{D}_1$, $\mathcal{C}_2$, $\mathcal{D}_2$ such that (i) intersection emptiness is decidable for $\mathcal{C}_1$ and $\mathcal{D}_1$, but regular separability is undecidable for $\mathcal{C}_1$ and $\mathcal{D}_1$ and (ii) regular separability is decidable for $\mathcal{C}_2$ and $\mathcal{D}_2$, but intersection emptiness is undecidable for $\mathcal{C}_2$ and $\mathcal{D}_2$.

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

Keywords and phrases Regular separability, intersection emptiness, decidability

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.51

Related Version `https://arxiv.org/pdf/1908.04038.pdf`

Acknowledgements We thank Lorenzo Clemente and Wojciech Czerwiński for fruitful discussions.

1 Introduction

The *intersection emptiness problem* for language classes $\mathcal{C}$ and $\mathcal{D}$ asks for two given languages K from $\mathcal{C}$ and L from $\mathcal{D}$, whether $K \cap L = \emptyset$. If $\mathcal{C}$ and $\mathcal{D}$ are language classes associated with classes of infinite-state systems, then intersection emptiness corresponds to verifying safety properties in concurrent systems where one system of $\mathcal{C}$ communicates with a system of $\mathcal{D}$ via messages or shared memory [6]. The question of separability is to decide whether two given languages are not only disjoint, but whether there exists a finite, easily verifiable, certificate for disjointness (and thus for safety). Specifically, the $\mathcal{S}$ *separability problem* for a fixed class $\mathcal{S}$ of separators and language classes $\mathcal{C}$ and $\mathcal{D}$ asks, for given languages K from $\mathcal{C}$ and L from $\mathcal{D}$, whether there exists a language $S \in \mathcal{S}$ with $K \subseteq S$ and $S \cap L = \emptyset$.

There is extensive literature dealing with the separability problem, with a range of different separators considered. One line of work concerns separability of regular languages by separators from a variety [1] of regular languages. Here, the investigation began with a more general problem, *computing pointlikes* (equivalently, the *covering problem*) [2, 20, 38], but

[1] By which we mean the algebraic notion.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 51; pp. 51:1–51:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

later also concentrated on separability (e.g. [32, 33, 34, 35, 36, 37]). Moreover, separability has been studied for regular tree languages, where separators are either piecewise testable tree languages [21] or languages of deterministic tree-walking automata [5]. For non-regular input languages, separability has been investigated with *piecewise testable languages* (PTL) [11] and generalizations thereof [42] as separators. Separability of subsets of trace monoids [7] and commutative monoids [9] by recognizable subsets has been studied as well.

A natural choice for the separators is the class of *regular languages*. On the one hand, they have relatively high separation power and on the other hand, it is usually verifiable whether a given regular language is in fact a separator. For instance, they generalize piecewise testable languages but are less powerful than context-free languages (CFL). Since the intersection problem for CFL is undecidable, it is not easy to check if a given candidate CFL is a separator.

This has motivated a recent research effort to understand for which language classes $\mathcal{C}, \mathcal{D}$ regular separability is decidable [29, 9, 8]. An early result was that regular separability is undecidable for CFL (by this we mean that both input languages are context-free) [39, 25]. This was recently strengthened to undecidability already for visibly pushdown languages [28] and one-counter languages [29]. On the positive side, it was shown that regular separability is decidable for several subclasses of vector addition systems (VASS): for one-dimensional VASS [29], for commutative VASS languages [9], and for Parikh automata (equivalently, $\mathbb{Z}$-VASS) [8]. Moreover, it is decidable for languages of well-structured transition systems [10]. Furthermore, decidability still holds in many of these cases if one of the inputs is a general VASS language [12]. However, if both inputs are VASS languages, decidability of regular separability remains a challenging open problem.

Of course, if one of the input languages is regular, checking regular separability degenerates into checking intersection with a regular language. Thus, the problem becomes interesting when both input languages are non-regular. Many language classes beyond the regular languages constitute *full trios*, meaning that they are closed under rational transductions. This is typically the case for classes that originate from non-deterministic infinite-state systems [16] and from various types of grammars [16, 13].

In the case of full trios, the available results exhibit a striking correspondence between regular separability and the intersection problem: Wherever decidability of regular separability has been clarified for a full trio, it is decidable if and only if intersection is decidable. Of the abovementioned languages classes, the context-free languages [3], languages of (one-dimensional) VASS [22], one-counter automata [3], Parikh automata [27], and well-structured transition systems [18] each constitute a full trio (visibly pushdown languages and commutative VASS languages do not form full trios). In fact, in the case of well-structured transition systems, it even turned out that two languages are regular-separable if and only if they are disjoint [10]. Moreover, deciding regular separability usually involves non-trivial refinements of the methods for deciding intersection. Without the restriction of being a full trio, there is an example of a language class where the intersection problem is decidable, but regular separability is not: the visibly pushdown languages for a fixed alphabet partition [28].

In light of these observations, there was a growing interest in whether there is a deeper connection between regular separability and intersection emptiness in the case of full trios. In other words: *Is regular separability just intersection emptiness in disguise?* It is conceivable that for full trios, regular separability and intersection emptiness are mutually reducible. An equivalence in this spirit already exists for separability by PTL: For full trios $\mathcal{C}$ and $\mathcal{D}$, separability by PTL for $\mathcal{C}$ and $\mathcal{D}$ is decidable if and only if the simultaneous unboundedness problem is decidable for $\mathcal{C}$ and for $\mathcal{D}$ [11]. These two problems, in turn, are equivalent to computing downward closures [41]. A further analogous equivalence is that full trios are closed under intersection if and only if they are closed under the shuffle operator [19].

Contribution. We show that regular separability and intersection emptiness are independent problems for full trios: Each problem can be decidable while the other is undecidable. Specifically, we present full trios $\mathcal{C}_1$, $\mathcal{D}_1$, $\mathcal{C}_2$, $\mathcal{D}_2$, so that (i) for $\mathcal{C}_1$ and $\mathcal{D}_1$, regular separability is undecidable, but intersection emptiness is decidable and (ii) for $\mathcal{C}_2$ and $\mathcal{D}_2$, regular separability is decidable, but intersection emptiness is undecidable. Some of these classes have been studied before (such as the higher-order pushdown languages), but some have not (to the best of our knowledge). However, they are all natural in the sense that they are defined in terms of machine models and have decidable emptiness and membership problems. We introduce two new classes defined by counter systems that accept based on certain numerical predicates. These predicates are specified either using reset vector addition systems or higher-order pushdown automata.

2 Preliminaries

We use Σ (sometimes Γ) to denote a finite set of letters and Σ^* to denote the set of finite strings (aka words) over the alphabet Σ. To distinguish between expressions over natural numbers and expressions involving words, we use typewriter font to denote letters, e.g. $\mathtt{a}$, $\mathtt{0}$, $\mathtt{1}$, etc. For example, $\mathtt{0}^n$ is the word consisting of an n-fold repetition of the letter $\mathtt{0}$, whereas 0^n is the number zero. The empty string is denoted ε. If $S \subseteq \mathbb{N}$ we write $\mathtt{a}^S$ for the set $\{\mathtt{a}^n \mid n \in S\} \subseteq \mathtt{a}^*$ and 2^S for the set $\{2^n \mid n \in S\} \subseteq \mathbb{N}$.

We define the map $\nu\colon \{\mathtt{0},\mathtt{1}\}^* \to \mathbb{N}$ which takes every word to the number which it denotes in binary representation: We define $\nu(\varepsilon) = 0$ and $\nu(w\mathtt{1}) = 2 \cdot \nu(w) + 1$ and $\nu(w\mathtt{0}) = 2 \cdot \nu(w)$ for $w \in \{\mathtt{0},\mathtt{1}\}^*$. For example, $\nu(\mathtt{110}) = 6$. Often we are only concerned with words of the form $\{\mathtt{0}\} \cup \mathtt{1}\{\mathtt{0},\mathtt{1}\}^*$. For subsets $L \subseteq \{\mathtt{0},\mathtt{1}\}^*$, we define $\nu(L) = \{\nu(w) \mid w \in L\}$.

Languages are denoted by L, L', K etc. and the language of a machine M is denoted by $\mathsf{L}(M)$. Classes of languages are denoted by $\mathcal{C}$, $\mathcal{D}$, etc.

▶ **Definition 2.1.** An *asynchronous transducer* $\mathcal{T}$ is a tuple $\mathcal{T} = (Q, \Gamma, \Sigma, E, q_0, F)$ with a set of finite states Q, finite output alphabet Γ, finite input alphabet Σ, a finite set of edges $E \subseteq Q \times \Gamma^* \times \Sigma^* \times Q$, initial state $q_0 \in Q$ and set of final states $F \subseteq Q$. We write $p \xrightarrow{v|u} q$ if $(p, v, u, q) \in E$ and the machine reads u in state p, outputs v and moves to state q. We also write $p \xrightarrow[*]{w|w'} q$ if there are states $q_0, q_1, \ldots, q_n$ and words $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n$ such that $p = q_0$, $q = q_n$, $w' = u_1u_2 \cdots u_n$, $w = v_1v_2 \cdots v_n$ and $q_i \xrightarrow{v_i|u_i} q_{i+1}$ for all $0 \leq i \leq n$.

The *transduction* $T \subseteq \Gamma^* \times \Sigma^*$ generated by the transducer $\mathcal{T}$ is the set of tuples $(v, u) \in \Gamma^* \times \Sigma^*$ such that $q_0 \xrightarrow[*]{v|u} q_f$ for some $q_f \in F$. Given a language $L \subseteq \Sigma^*$, we define $TL := \{v \in \Gamma^* \mid \exists u \in L\ (v, u) \in T\}$. A transduction $T \subseteq \Gamma^* \times \Sigma^*$ is *rational* if it is generated by some asynchronous transducer.

A *language* is a subset of Γ^* for some alphabet Γ. A *language class* is a collection of languages, together with some way to finitely represent these languages, for example using machine models or grammars. We call a language class a *full trio* if it is effectively closed under rational transductions. This means, given a representation of L in $\mathcal{C}$ and an asynchronous transducer for $T \subseteq \Gamma^* \times \Sigma^*$, the language TL belongs to $\mathcal{C}$ and one can compute a representation of TL in $\mathcal{C}$.

The following equivalent definition of full trios is well known (see Berstel [3]):

▶ **Lemma 2.2.** *A language class is closed under rational transductions if and only if it is effectively closed under (i) homomorphic image, (ii) inverse homomorphic image, and (iii) intersection with regular languages.*

We are interested in decision problems where the representation of a language L (or possibly multiple languages) is the input. In particular, we study the following problems.

▶ **Problem 2.3** (Intersection Emptiness)**.** For languages classes $\mathcal{C}_1$ and $\mathcal{C}_2$, the *intersection emptiness problem*, briefly $\mathsf{IE}(\mathcal{C}_1, \mathcal{C}_2)$, is defined as follows:
Input: Languages L_1 from $\mathcal{C}_1$ and L_2 from $\mathcal{C}_2$.
Question: Is $L_1 \cap L_2$ empty?

▶ **Problem 2.4** (Regular Separability)**.** For languages classes $\mathcal{C}_1$ and $\mathcal{C}_2$, the *regular separability problem*, briefly $\mathsf{RS}(\mathcal{C}_1, \mathcal{C}_2)$, is defined as follows:
Input: Languages L_1 from $\mathcal{C}_1$ and L_2 from $\mathcal{C}_2$.
Question: Is there a regular language R such that $L_1 \subseteq R$ and $L_2 \cap R = \emptyset$?
We will write $L|K$ to denote that L and K are regular-separable.

▶ **Problem 2.5** (Emptiness)**.** The *emptiness problem* for a language class $\mathcal{C}$, briefly $\mathsf{Empty}(\mathcal{C})$, is defined as:
Input: A language L from $\mathcal{C}$.
Question: Is $L = \emptyset$, i.e. is L empty?

▶ **Problem 2.6** (Infinity)**.** The *infinity problem* for a language class $\mathcal{C}$, briefly $\mathsf{Inf}(\mathcal{C})$, is defined as:
Input: A language L from $\mathcal{C}$.
Question: Does L contain infinitely many words?

3 Incrementing automata

The counterexamples we construct are defined using special kinds of automata that can only increment a counter, which we will define formally below. The acceptance condition requires that the counter value satisfies a specific *numerical predicate*, in addition to reaching a final state. By a *predicate class*, we mean a class $\mathcal{P}$ of predicates over natural numbers (i.e. subsets $P \subseteq \mathbb{N}$) such that there is a way to finitely describe the members of $\mathcal{P}$. As an example, if $\mathcal{C}$ is a language class, then a subset $S \subseteq \mathbb{N}$ is a *pseudo-$\mathcal{C}$ predicate* if $S = \nu(L)$ for some $L \in \mathcal{C}$ and $L \subseteq \{\mathtt{0}, \mathtt{1}\}^*$. Now the class of all pseudo-$\mathcal{C}$ predicates constitutes a predicate class, because a pseudo-$\mathcal{C}$ predicate can be described using the finite description of a language in $\mathcal{C}$. The class of all pseudo-$\mathcal{C}$ predicates is denoted $\mathsf{pseudo}\mathcal{C}$.

▶ **Definition 3.1.** Let $\mathcal{P}$ be a predicate class. An *incrementing automaton over* $\mathcal{P}$ is a five-tuple $\mathcal{M} = (Q, \Sigma, E, q_0, F)$ where Q is a finite set of *states*, Σ is its *input alphabet*, $E \subseteq Q \times \Sigma^* \times \{0, 1\} \times Q$ a finite set of *edges*, $q_0 \in Q$ an initial state and F is a finite set of *acceptance pairs* (q, P) where $q \in Q$ is a state and P belongs to $\mathcal{P}$.

A *configuration* of $\mathcal{M}$ is a pair $(q, n) \in Q \times \mathbb{N}$. For two configurations (q, n), (q', n'), we write $(q, n) \xrightarrow{w} (q', n')$ if there are configurations $(q_1, n_1), \dots, (q_\ell, n_\ell)$ with $q_1 = q$ and $q_\ell = q'$ and edges (q_i, w_i, m_i, q_{i+1}) with $n_{i+1} = n_i + m_i$ for $1 \le i < \ell$ and $w = w_1 \cdots w_\ell$. The *language accepted by* $\mathcal{M}$ is

$$\mathsf{L}(\mathcal{M}) = \{w \in \Sigma^* \mid (q_0, 0) \xrightarrow{w} (q, m) \text{ for some } (q, P) \text{ in } F \text{ with } m \in P\}.$$

The collection of all languages accepted by incrementing automata over $\mathcal{P}$ is denoted $\mathcal{I}(\mathcal{P})$.

It turns out that even with no further assumptions on the predicate class $\mathcal{P}$, the language class $\mathcal{I}(\mathcal{P})$ has some nice closure properties.

▶ **Lemma 3.2.** *Let $\mathcal{P}$ be a predicate class. The languages of incrementing automata over $\mathcal{P}$ are precisely the finite unions of languages of the form $T\mathtt{a}^P$ where $P \in \mathcal{P}$ and $T \subseteq \Sigma^* \times \{\mathtt{a}\}^*$ is a rational transduction. In particular, the class of languages accepted by incrementing automata over $\mathcal{P}$ is a full trio.*

Proof. For every accepting pair (q, P) of $\mathcal{M}$, we construct a transducer $T_{q,P}$, which has the same set of states as $\mathcal{M}$, accepting state set $\{q\}$ and for each edge (q', w, m, q'') of $\mathcal{M}$ the transducer reads $\mathtt{a}$ if $m = 1$ or ε if $m = 0$ and outputs w. Then $\mathsf{L}(\mathcal{M})$ is the finite union of all $T_{q,P}(\mathtt{a}^P)$.

Conversely, since the languages accepted by incrementing automata over $\mathcal{P}$ are clearly closed under union, it suffices to show that $T\mathtt{a}^P$ is accepted by an incrementing automaton over $\mathcal{P}$. We may assume that T is given by a transducer in which every edge is of the form $(q, w, \mathtt{a}^m, q')$ with $m \in \{0, 1\}$. Let $\mathcal{M}$ have the same state set as T and turn every edge $(q, w, \mathtt{a}^m, q')$ into an edge (q, w, m, q') for $\mathcal{M}$. Finally, for every final state q of T, we give $\mathcal{M}$ an accepting pair (q, P). Then clearly $\mathsf{L}(\mathcal{M}) = T\mathtt{a}^P$.

This implies that the class of incrementing automata over $\mathcal{P}$ is a full trio: If $L \subseteq \Sigma^*$ is accepted by a incrementing automata over $\mathcal{P}$, then we can write $L = T_1\mathtt{a}^{P_1} \cup \cdots \cup T_\ell\mathtt{a}^{P_\ell}$ with $T_1, \ldots, T_\ell \subseteq \Sigma^* \times \mathtt{a}^*$. If $T \subseteq \Gamma^* \times \Sigma^*$ is a rational transduction, then $TL = (TT_1)\mathtt{a}^{P_1} \cup \cdots \cup (TT_\ell)\mathtt{a}^{P_\ell}$ and since TT_i is again a rational transduction for $1 \leq i \leq \ell$, the lanuage TL is accepted by some incrementing automaton over $\mathcal{P}$. ◀

It is obvious that the class $\mathcal{I}(\mathcal{P})$ does not always have a decidable emptiness problem: Emptiness is decidable for $\mathcal{I}(\mathcal{P})$ if and only if it is decidable whether a given predicate from $\mathcal{P}$ intersects a given arithmetic progression, i.e. given P and $m, n \in \mathbb{N}$, whether $(m+n\mathbb{N}) \cap P \neq \emptyset$. For all the predicate classes $\mathcal{P}$ we consider, emptiness for $\mathcal{I}(\mathcal{P})$ will always be decidable.

4 Decidable Intersection and Undecidable Regular Separability

In this section, we present a language class $\mathcal{C}$ so that the intersection emptiness problem $\mathsf{IE}(\mathcal{C}, \mathcal{C})$ is decidable for $\mathcal{C}$, but the regular separability problem $\mathsf{RS}(\mathcal{C}, \mathcal{C})$ is undecidable for $\mathcal{C}$. The definition of $\mathcal{C}$ is based on reset vector addition systems.

Reset Vector Addition Systems. A *reset vector addition system* (reset VASS) is a tuple $\mathcal{V} = (Q, \Sigma, n, E, q_0, F)$, where Q is a finite set of *states*, Σ is its *finite input alphabet*, $n \in \mathbb{N}$ is its number of counters, $E \subseteq Q \times \Sigma^* \times \{1, \ldots, n\} \times \{0, 1, -1, \mathsf{r}\} \times Q$ is a finite set of *edges*, $q_0 \in Q$ is its *initial state*, and $F \subseteq Q$ is its set of *final states*. A *configuration* of $\mathcal{V}$ is a tuple $(q, m_1, \ldots, m_n)$ where $q \in Q$ and $m_1, \ldots, m_n \in \mathbb{N}$. We write $(q, m_1, \ldots, m_n) \xrightarrow{w} (q', m'_1, \ldots, m'_n)$ if there is an edge (q, w, k, x, q') such that for every $j \neq k$, we have $m'_j = m_j$ and

- if $x \in \{-1, 0, 1\}$, then $m'_k = m_k + x$,
- if $x = \mathsf{r}$, then $m'_k = 0$.

If there are configurations $c_1, \ldots, c_\ell$ and words $w_1, \ldots, w_{\ell-1}$ with $c_i \xrightarrow{w_i} c_{i+1}$ for $1 \leq i < \ell$, and $w = w_1 \cdots w_\ell$, then we also write $c_1 \xrightarrow{w} c_\ell$. The *language accepted by* $\mathcal{V}$ is defined as

$$\mathsf{L}(\mathcal{V}) = \{w \in \Sigma^* \mid (q_0, 0, \ldots, 0) \xrightarrow{w} (q, m_1, \ldots, m_n) \text{ for some } q \in F \text{ and } m_1, \ldots, m_n \in \mathbb{N}\}.$$

The class of languages accepted by reset VASS is denoted $\mathcal{R}$.

Our language class will be $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$, i.e. incrementing automata with access to predicates of the form $\nu(L)$ where $L \subseteq \{\mathtt{0}, \mathtt{1}\}^*$ is the language of a reset VASS.

▶ **Theorem 4.1.** $\mathsf{RS}(\mathcal{I}(\mathsf{pseudo}\mathcal{R}), \mathcal{I}(\mathsf{pseudo}\mathcal{R}))$ *is undecidable and* $\mathsf{IE}(\mathcal{I}(\mathsf{pseudo}\mathcal{R}), \mathcal{I}(\mathsf{pseudo}\mathcal{R}))$ *is decidable.*

Note that $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$ is a full trio (Lemma 3.2) and since intersection is decidable, in particular its emptiness problem is decidable: For $L \subseteq \Sigma^*$, one has $L \cap \Sigma^* = \emptyset$ if and only if $L = \emptyset$. Moreover, note that we could not have chosen $\mathcal{R}$ as our example class: Since reset VASS are well-structured transition systems, regular separability is decidable for them [10].

Before we begin with the proof of Theorem 4.1, let us mention that instead of $\mathcal{R}$, we could have chosen any language class $\mathcal{D}$, for which (i) $\mathcal{D}$ is closed under rational transductions, (ii) $\mathcal{D}$ is closed under intersection, (iii) $\mathsf{Empty}(\mathcal{D})$ is decidable and (iv) $\mathsf{Inf}(\mathcal{D})$ is undecidable. For example, we could have also used lossy channel systems instead of reset VASS.

We now recall some results regarding $\mathcal{R}$ from literature.

▶ **Lemma 4.2.** *Emptiness is decidable for* $\mathcal{R}$.

The lemma follows from the fact that reset VASS are well-structured transition systems [14], for which the coverability problem is decidable [1, 17] and the fact that a reset VASS has a non-empty language if and only if a particular configuration is coverable.

The following can be shown using standard product constructions, please see the full version [40].

▶ **Lemma 4.3.** $\mathcal{R}$ *is closed under rational transductions, union, and intersection.*

We now show that regular separability is undecidable for $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$. We do this using a reduction from the infinity problem for $\mathcal{R}$, whose undecidability is an easy consequence of the undecidability of boundedness of reset VASS.

The boundedness problem for reset VASS is defined below and was shown to be undecidable by Dufourd, Finkel, and Schnoebelen [14] (and a simple and more general proof was given by Mayr [30]). A configuration $(q, x_1, \ldots, x_n)$ is *reachable* if there is a $w \in \Sigma^*$ with $(q_0, 0, \ldots, 0) \xrightarrow{w} (q, x_1, \ldots, x_n)$. A reset VASS $\mathcal{V}$ is called *bounded* if there is a $B \in \mathbb{N}$ such that for every reachable $(q, x_1, \ldots, x_n)$, we have $x_1 + \cdots + x_n \leq B$. Hence, the *boundedness problem* is the following.

Input: A reset VASS $\mathcal{V}$.
Question: Is $\mathcal{V}$ bounded?

▶ **Lemma 4.4.** *The infinity problem for* $\mathcal{R}$ *is undecidable.*

Proof. From an input reset VASS $\mathcal{V} = (Q, \Sigma, n, E, q_0, F)$, we construct a reset VASS $\mathcal{V}'$ over the alphabet $\Sigma' = \{\mathtt{a}\}$ as follows. In every edge of $\mathcal{V}$, we replace the input word by the empty word ε. Moreover, we add a fresh state s, which is the only final state of $\mathcal{V}'$. Then, we add an edge $(q, \varepsilon, 1, 0, s)$ for every state q of $\mathcal{V}$. Finally, we add a loop $(s, \mathtt{a}, i, -1, s)$ for every $i \in \{1, \ldots, n\}$. This means $\mathcal{V}'$ simulates a computation of $\mathcal{V}$ (but disregarding the input) and can spontaneously jump into the state s, from where it can decrement counters. Each time it decrements a counter in s, it reads an $\mathtt{a}$ from the input. Thus, clearly, $\mathsf{L}(\mathcal{V}') \subseteq \mathtt{a}^*$. Moreover, we have $\mathtt{a}^m \in \mathsf{L}(\mathcal{V}')$ if and only if there is a reachable configuration $(q, x_1, \ldots, x_n)$ of $\mathcal{V}$ with $x_1 + \cdots + x_n \geq m$. Thus, $\mathsf{L}(\mathcal{V}')$ is finite if and only if $\mathcal{V}$ is bounded. ◀

Note that infinity is already undecidable for languages that are subsets of $\mathtt{10}^*$. This is because given L from $\mathcal{R}$, a rational transduction yields $L' = \{\mathtt{1}\mathtt{0}^{|w|} \mid w \in L\}$ and L' is infinite if and only if L is.

Our reduction from the infinity problem works because the input languages have a particular shape, for which regular separability has a simple characterization.

▶ **Lemma 4.5.** *Let $S_0, S_1 \subseteq \mathbb{N}$ and $\mathbb{N} \setminus 2^{\mathbb{N}} \subseteq S_1$. Then $\mathtt{a}^{S_0}$ and $\mathtt{a}^{S_1}$ are regular-separable if and only if S_0 is finite and disjoint from S_1.*

Proof. If S_0 is finite and disjoint from S_1, then clearly $\mathtt{a}^{S_0}$ is a regular separator. For the "only if" direction, consider any infinite regular language $R \subseteq \mathtt{a}^*$. It has to include an arithmetic progession, meaning that there exist $m, n \in \mathbb{N}$ with $\mathtt{a}^{m+n\mathbb{N}} \subseteq R$. Hence, for sufficiently large ℓ, the language $\{\mathtt{a}^x \mid 2^\ell < x < 2^{\ell+1}\} \subseteq S_1$ must intersect with R. In other words, no infinite R can be a regular separator of $\mathtt{a}^{S_0}$ and $\mathtt{a}^{S_1}$ i.e. S_0 must be finite (and disjoint from S_1). ◀

▶ **Lemma 4.6.** *Regular separability is undecidable for $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$.*

Proof. We reduce the infinity problem for $\mathcal{R}$ (which is undecidable by Lemma 4.4) to regular separability in $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$. Suppose we are given L from $\mathcal{R}$. Since $\mathcal{R}$ is effectively closed under rational transductions, we also have $K = \{\mathtt{1}\mathtt{0}^{|w|} \mid w \in L\}$ in $\mathcal{R}$. Note that K is infinite if and only if L is infinite. Then $\nu(K) \subseteq 2^{\mathbb{N}}$ and $K_1 := \mathtt{a}^{\nu(K)}$ belongs to $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$. Let $K_2 = \mathtt{a}^{\mathbb{N}\setminus 2^{\mathbb{N}}} = \mathtt{a}^{\nu(\mathtt{1}\{\mathtt{0},\mathtt{1}\}^*\mathtt{1}\{\mathtt{0},\mathtt{1}\}^*)}$, which also belongs to $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$, because $\mathtt{1}\{\mathtt{0},\mathtt{1}\}^*\mathtt{1}\{\mathtt{0},\mathtt{1}\}^*$ is regular and thus a member of $\mathcal{R}$.

By Lemma 4.4, K_1 and K_2 are regular-separable if and only if K_1 is finite and disjoint from K_2. Since $K_1 \cap K_2 = \emptyset$ by construction, we have regular separability if and only if K_1 is finite, which happens if and only if K is finite. ◀

For Theorem 4.1, it remains to show that intersection is decidable for $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$. We do this by expressing intersection non-emptiness in the logic $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$, which is the positive Σ_1 fragment of Presburger arithmetic extended with pseudo-$\mathcal{R}$ predicates. Moreover, we show that this logic has a decidable truth problem.

We begin with some notions from first-order logic (please see [15] for syntax and semantics of first-order logic). First-order formulae will be denoted by $\phi(\bar{x}), \psi(y)$ etc. where $\bar{x}$ is a tuple of (possibly superset of the) free variables and y is a single free variable. For a formula $\phi(\bar{x})$, we denote by $[\![\phi(\bar{x})]\!]$ the set of its solutions (in our case, the domain is $\mathbb{N}$).

Our decision procedure for $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ is essentially the same as the procedure to decide the first-order theory of automatic structures [4], except that instead of regular languages, we use $\mathcal{R}$. For $\bar{w} = (w_1, w_2, \ldots, w_k) \in (\Sigma^*)^k$, the *convolution* $w_1 \otimes w_2 \otimes \ldots \otimes w_k$ is a word over the alphabet $(\Sigma \cup \{\Box\})^k$ where $\Box$ is a padding symbol not present in Σ. If $w_i = w_{i1}w_{i2}\ldots w_{im_i}$ and $m = \max\{m_1, m_2, \ldots, m_k\}$ then

$$w_1 \otimes w_2 \otimes \ldots \otimes w_k := \begin{bmatrix} w'_{11} \\ w'_{21} \\ \vdots \\ w'_{k1} \end{bmatrix} \cdots \begin{bmatrix} w'_{1m} \\ w'_{2m} \\ \vdots \\ w'_{km} \end{bmatrix} \in ((\Sigma \cup \{\Box\})^k)^*$$

where $w'_{i1} \cdots w'_{im} = \Box^{m-m_i} w_i$ for $1 \leq i \leq k$. We say that a k-ary (arithmetic) relation $R \subseteq \mathbb{N}^k$ is a *pseudo-$\mathcal{R}$ relation* if the set of words $L_R = \{w_1 \otimes w_2 \otimes \cdots \otimes w_k \mid (\nu(w_1), \ldots, \nu(w_k)) \in R\}$ belongs to $\mathcal{R}$. In our decision procedure for $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$, we will show inductively that every formula defines a pseudo-$\mathcal{R}$ relation.

▶ Remark 4.7. Note that our definition of the convolution deviates from the usual one that pads words on the right [4, 26]. This is because we want pseudo-$\mathcal{R}$ predicates to be pseudo-$\mathcal{R}$ relations. By our definition of ν, this means the least significant bit will always be on the right. Since we also want the ternay addition relation $\{(x, y, z) \in \mathbb{N}^3 \mid x + y = z\}$ to be a pseudo-$\mathcal{R}$ relation, we need to align the words in the convolution at the least significant bit and thus pad on the left.

Formally, we consider the theory $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ where $(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ is the structure with domain $\mathbb{N}$ of natural numbers, the constant symbol 1 and the binary symbols $+$ and $\leq$ taking their canonical interpretations and $\mathsf{pseudo}\mathcal{R}$ is a set of predicate symbols, one for each pseudo-$\mathcal{R}$ predicate. By Σ_1^+ we mean the fragment of first order formulae obtained by using only the boolean operations $\wedge, \vee$ and existential quantification.

▶ **Definition 4.8.** Let $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ be the set of first order logic formulae given by the following grammar:

$$\phi(\bar{x}, \bar{y}, \bar{z}) := S(x) \mid t_1 \leq t_2 \mid \phi_1(\bar{x}, \bar{y}) \wedge \phi_2(\bar{x}, \bar{z}) \mid \phi_1(\bar{x}, \bar{y}) \vee \phi_2(\bar{x}, \bar{z}) \mid \exists y\ \phi'(y, \bar{x})$$

where S is from $\mathsf{pseudo}\mathcal{R}$ and t_1, t_2 are terms obtained from using variables, 1 and $+$.

▶ **Lemma 4.9.** *The truth problem for* $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ *is decidable.*

Proof. It is clear that by introducing new existentially quantified variables, one can transform each formula from $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ into an equivalent formula that is generated by the simpler grammar

$$\begin{aligned} \phi(\bar{x}, \bar{y}, \bar{z}) := & S(x) \mid x + y = z \mid x = 1 \mid \\ & \phi_1(\bar{x}, \bar{y}) \wedge \phi_2(\bar{x}, \bar{z}) \mid \phi_1(\bar{x}, \bar{y}) \vee \phi_2(\bar{x}, \bar{z}) \mid \exists y\ \phi'(y, \bar{x}) \end{aligned}$$

We want to show that given any input sentence ψ from $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$, we can decide if it is true or not. If the sentence has no variables, then it is trivial to decide. Otherwise, $\psi = \exists \bar{x}\ \phi(\bar{x})$ for some formula $\phi(\bar{x})$. We claim that the solution set $R = [\![\phi(\bar{x})]\!]$ is a pseudo-$\mathcal{R}$ relation and a reset VASS for L_R can be effectively computed. Assuming the claim, the truth of ψ reduces to the emptiness of $[\![\phi(\bar{x})]\!]$ or equivalently the emptiness of L_R, which is decidable by Lemma 4.2.

We prove the claim by structural induction on the defining formula $\phi(\bar{x})$, please see the full version [40] for details. ◀

▶ Remark 4.10. The truth problem for $\Pi_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ is undecidable by reduction from the infinity problem for $\mathcal{R}$. Given $L \subseteq \mathtt{10}^*$, let $R_L = \nu(L) \subseteq \mathbb{N}$ be the predicate corresponding to L. Now the downward closure $D := \{x \in \mathbb{N} \mid \exists y \colon x \leq y \wedge R_L(y)\}$ is definable in $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ and therefore $K := \nu(D)$ belongs to $\mathcal{R}$ by the proof of Lemma 4.9. Then the Π_1^+-sentence $\forall x \colon R_K(x)$ is true if and only if L is infinite.

Having established that $\Sigma_1^+(\mathbb{N}, +, \leq, 1, \mathsf{pseudo}\mathcal{R})$ is decidable, we are ready to show that intersection emptiness is decidable for $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$.

▶ **Lemma 4.11.** *The intersection problem is decidable for* $\mathcal{I}(\mathsf{pseudo}\mathcal{R})$.

Proof. Given $L_1, L_2 \in \mathcal{I}(\mathsf{pseudo}\mathcal{R})$, by Lemma 3.2, we know that both L_1 and L_2 are finite unions of languages of the form $T\mathtt{a}^S$, where S is a pseudo-$\mathcal{R}$ predicate. Therefore, it suffices to decide the emptiness of intersections of the form $T_1\mathtt{a}^{S_1} \cap T_2\mathtt{a}^{S_2}$ where S_1 and S_2 are pseudo-$\mathcal{R}$ predicates. Note that $T_1\mathtt{a}^{S_1} \cap T_2\mathtt{a}^{S_2} = \emptyset$ iff $T_2^{-1}T_1\mathtt{a}^{S_1} \cap \mathtt{a}^{S_2} = \emptyset$. Since $T_2^{-1}T_1$ is again a rational transduction, it suffices to check emptiness of languages of the form $T\mathtt{a}^{S_1} \cap \mathtt{a}^{S_2}$ where $T \subseteq \mathtt{a}^* \times \mathtt{a}^*$ is a rational transduction. Notice that we can construct an automaton $\mathcal{A}$ over the alphabet $\Sigma' = \{\mathtt{b}, \mathtt{c}\}$ with the same states as the transducer $\mathcal{M}_T$ for T and where for any transition $p \xrightarrow{\mathtt{a}^m|\mathtt{a}^n} q$ of $\mathcal{M}_T$ we have a transition $p \xrightarrow{\mathtt{b}^m\mathtt{c}^n} q$ in $\mathcal{A}$. It is clear that $(\mathtt{a}^x, \mathtt{a}^y) \in T$ iff there exists a word $w \in \mathsf{L}(\mathcal{A})$ such that w contains exactly x

occurrences of $\mathtt{b}$ and y occurrences of $\mathtt{c}$. Now it follows from Parikh's theorem [31] that the set $\{(x,y)\in\mathbb{N}\times\mathbb{N}\mid(\mathtt{a}^x,\mathtt{a}^y)\in T\}$ is semilinear, meaning that there are numbers $n_0,\ldots,n_k$ and $m_0,\ldots,m_k$ such that $(\mathtt{a}^x,\mathtt{a}^y)\in T$ if and only if

$$\exists z_1\exists z_2\ldots\exists z_k\ (x=n_0+\sum_{i=i}^{k}z_in_i)\ \wedge\ (y=m_0+\sum_{i=i}^{k}z_im_i).$$

In particular, there is a formula $\phi_T(x,y)$ in $\Sigma_1^+(\mathbb{N},+,\leq,1,\mathsf{pseudo}\mathcal{R})$ such that $(\mathtt{a}^x,\mathtt{a}^y)\in T$ if and only if $\phi_T(x,y)$ is satisfied. We can now write a formula $\phi_2(x)$ in $\Sigma_1^+(\mathbb{N},+,\leq,1,\mathsf{pseudo}\mathcal{R})$ such that $\phi_2(x)$ is satisfied if and only if $\mathtt{a}^x\in T\mathtt{a}^{S_2}$:

$$\phi_2(x):=\exists y\ \big(\phi_T(x,y)\ \wedge\ S_2(y)\big)$$

In the same way, the formula $\phi_1(x):=S_1(x)$ defines $\mathtt{a}^{S_1}$. Now set $\phi=\exists x\ \big(\phi_1(x)\ \wedge\ \phi_2(x)\big)$. Then ϕ is true if and only if $T\mathtt{a}^{S_2}\cap\mathtt{a}^{S_1}\neq\emptyset$. Decidability of $\mathsf{IE}(\mathcal{I}(\mathsf{pseudo}\mathcal{R}),\mathcal{I}(\mathsf{pseudo}\mathcal{R}))$ follows from Lemma 4.9. ◀

5 Decidable Regular Separability and Undecidable Intersection

In this section, we present language classes $\mathcal{C}$ and $\mathcal{D}$ so that $\mathsf{IE}(\mathcal{C},\mathcal{D})$ is undecidable, but $\mathsf{RS}(\mathcal{C},\mathcal{D})$ is decidable. These classes are constructed using higher-order pushdown automata, which we define first.

We follow the definition of [23]. Higher-order pushdown automata are a generalization of pushdown automata where instead of manipulating a stack, one can manipulate a stack of stacks (order-2), a stack of stacks of stacks (order-3), etc. Therefore, we begin by defining these higher-order stacks. While for ordinary (i.e. order-1) pushdown automata, stacks are words over the stack alphabet Γ, order-$(k+1)$ stacks are sequences of order-k stacks. Let Γ be an alphabet and $k\in\mathbb{N}$. The set of *order-k stacks* $\mathcal{S}_k^\Gamma$ is inductively defined as follows:

$$\mathcal{S}_0^\Gamma=\Gamma,\qquad \mathcal{S}_{k+1}^\Gamma=\{[s_1\cdots s_m]_{k+1}\mid m\geq 1,\ s_1,\ldots,s_m\in\mathcal{S}_k^\Gamma\}.$$

For a word $v\in\Gamma^+$, the stack $[\cdots[[v]_1]_2\cdots]_k$ is also denoted $[\![v]\!]_k$. The function top yields the topmost symbol from Γ. This means, we have $\mathsf{top}([s_1\cdots s_m]_1)=s_m$ and $\mathsf{top}([s_1\cdots s_m]_k)=\mathsf{top}(s_m)$ for $k>1$.

Higher-order pushdown automata operate on higher-order stacks by way of instructions. For the stack alphabet Γ and for order-k stacks, we have the instruction set $I_k^\Gamma=\{\mathsf{push}_i,\mathsf{pop}_i\mid 1\leq i\leq k\}\cup\{\mathsf{rew}_\gamma\mid\gamma\in\Gamma\}$. These instructions act on $\mathcal{S}_k^\Gamma$ as follows:

$$\begin{array}{lll}
[s_1\cdots s_m]_1\cdot\mathsf{rew}_\gamma & =[s_1\cdots s_{m-1}\gamma]_1 & \\
[s_1\cdots s_m]_k\cdot\mathsf{rew}_\gamma & =[s_1\cdots s_{m-1}(s_m\cdot\mathsf{rew}_\gamma)]_k & \text{if } k>1\\
[s_1\cdots s_m]_i\cdot\mathsf{push}_i & =[s_1\cdots s_ms_m]_i & \\
[s_1\cdots s_m]_k\cdot\mathsf{push}_i & =[s_1\cdots s_m\ (s_m\cdot\mathsf{push}_i)]_k & \text{if } k>i\\
[s_1\cdots s_m]_i\cdot\mathsf{pop}_i & =[s_1\cdots s_{m-1}]_i & \text{if } m\geq 2\\
[s_1\cdots s_m]_k\cdot\mathsf{pop}_i & =[s_1\cdots s_{m-1}\ (s_m\cdot\mathsf{pop}_i)]_k & \text{if } k>i
\end{array}$$

and in all other cases, the result is undefined. For a word $w\in(I_k^\Gamma)^*$ and a stack $s\in\mathcal{S}_k^\Gamma$, the stack $s\cdot w$ is defined inductively by $s\cdot\varepsilon=s$ and $s\cdot(wx)=(s\cdot w)\cdot x$ for $x\in I_k^\Gamma$.

An *(order-k) higher-order pushdown automaton* (short HOPA) is a tuple $\mathcal{A}=(Q,\Sigma,\Gamma,\bot,E,q_0,F)$, where Q is a finite set of *states*, Σ is its *input alphabet*, Γ is its *stack alphabet*, $\bot\in\Gamma$ is its *stack bottom symbol*, E is a finite subset of $Q\times\Sigma^*\times\Gamma\times(I_k^\Gamma)^*\times Q$

whose elements are called *edges*, $q_0 \in Q$ is its *initial state*, and $F \subseteq Q$ is its set of *final states*. A *configuration* is a pair $(q, s) \in Q \times \mathcal{S}_k^\Gamma$. When drawing a higher-order pushdown automaton, an edge (q, u, γ, v, q') is represented by an arc $q \xrightarrow{u|\gamma|v} q'$. An arc $q \xrightarrow{u|v} q'$ means that for each $\gamma \in \Gamma$, there is an edge (q, u, γ, v, q').

For configurations $(q, s), (q', s')$ and a word $u \in \Sigma^*$, we write $(q, s) \xrightarrow{u}_{\mathcal{A}} (q', s')$ if there are edges $(q_1, u_1, \gamma_1, v_1, q_2), (q_2, u_2, \gamma_2, v_2, q_3), \ldots, (q_{n-1}, u_{n-1}, \gamma_{n-1}, v_{n-1}, q_n)$ in E and stacks $s_1, \ldots, s_n \in \mathcal{S}_k^\Gamma$ with $\mathsf{top}(s_i) = \gamma_i$ and $s_i \cdot v_i = s_{i+1}$ for $1 \leq i \leq n-1$ such that $(q, s) = (q_1, s_1)$ and $(q', s') = (q_n, s_n)$ and $u = u_1 \cdots u_n$. The *language accepted by* $\mathcal{A}$ is defined as

$$\mathsf{L}(\mathcal{A}) = \{w \in \Sigma^* \mid (q_0, [\![\bot]\!]_k) \xrightarrow{w}_{\mathcal{A}} (q, s) \text{ for some } q \in F \text{ and } s \in \mathcal{S}_k^\Gamma\}.$$

The languages accepted by order-k pushdown automata are called *order-k pushdown languages*. By $\mathcal{H}$, we denote the class of languages accepted by an order-k pushdown automaton for some $k \in \mathbb{N}$. In our example of classes with decidable regular separability and undecidable intersection, one of the two classes is $\mathcal{H}$. The other class will again be defined using incrementing automata.

▶ **Definition 5.1.** Let $\mathcal{C}$ be a language class. A predicate $P \subseteq \mathbb{N}$ is a *power-$\mathcal{C}$ predicate* if $P = \mathbb{N} \setminus 2^{\mathbb{N}} \ \cup \ \{2^{\nu(w)} \mid w \in L\}$ for some language L from $\mathcal{C}$. The class of power-$\mathcal{C}$ predicates is denoted $\mathsf{power}\mathcal{C}$.

Our example of classes with decidable regular separability but undecidable intersection is $\mathcal{H}$ on the one hand and $\mathcal{I}(\mathsf{power}\mathcal{H})$ on the other hand. It is well-known that $\mathcal{H}$ is a full trio (see, e.g., [16]). Moreover, $\mathcal{I}(\mathsf{power}\mathcal{H})$ is a full trio according to Lemma 3.2.

▶ **Theorem 5.2.** $\mathsf{RS}(\mathcal{H}, \mathcal{I}(\mathsf{power}\mathcal{H}))$ *is decidable, whereas* $\mathsf{IE}(\mathcal{H}, \mathcal{I}(\mathsf{power}\mathcal{H}))$ *is undecidable.*

Note that decidable regular separability implies that $\mathcal{I}(\mathsf{power}\mathcal{H})$ has a decidable emptiness problem: For $L \subseteq \Sigma^*$, one has $\Sigma^* | L$ if and only if $L = \emptyset$. Moreover, note that we could not have chosen $\mathcal{H}$ as our counterexample, because regular separability is undecidable for $\mathcal{H}$ (already for context-free languages) [39, 25].

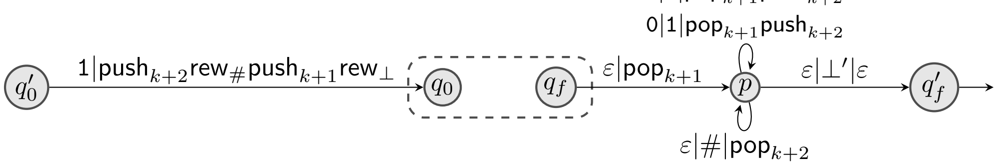
For showing Theorem 5.2, we rely on two ingredients. The first is that infinity is decidable for higher-order pushdown languages. This is a direct consequence of a result of Hague, Kochems and Ong [23], showing that the more general simultaneous unboundedness problem [41] and diagonal problem [11] are decidable for higher-order pushdown automata.

▶ **Lemma 5.3** **([23]).** $\mathsf{Inf}(\mathcal{H})$ *is decidable.*

The other ingredient is that turning binary representations into unary ones can be achieved in higher-order pushdown automata.

▶ **Lemma 5.4.** *If $L \subseteq \{\mathsf{0}, \mathsf{1}\}^*$ is an order-k pushdown language, then $L' = \{\mathsf{1}\mathsf{0}^{\nu(w)} \mid w \in L\}$ is an order-$(k+2)$ pushdown language.*

Proof. Let $\mathcal{A}$ be an order-k HOPA accepting $L \subseteq \{\mathsf{0}, \mathsf{1}\}^*$. We construct an order-$(k+2)$ HOPA $\mathcal{A}'$ for L'. We may clearly assume that $\mathcal{A}$ has only one final state q_f. The following diagram describes $\mathcal{A}'$:


$\varepsilon|\mathsf{0}|\mathsf{pop}_{k+1}\mathsf{push}_{k+2}$
$\mathsf{0}|\mathsf{1}|\mathsf{pop}_{k+1}\mathsf{push}_{k+2}$
q_0'
$\mathsf{1}|\mathsf{push}_{k+2}\mathsf{rew}_{\#}\mathsf{push}_{k+1}\mathsf{rew}_{\bot}$
q_0
q_f
$\varepsilon|\mathsf{pop}_{k+1}$
p
$\varepsilon|\bot'|\varepsilon$
q_f'
$\varepsilon|\#|\mathsf{pop}_{k+2}$

The HOPA $\mathcal{A}'$ starts in the configuration $(q_0', [\![\bot']\!]_{k+2})$ and in moving to q_0, it reads $\mathtt{1}$ and goes to $(q_0, [[\![\bot']\!]_{k+1}[[\![\#]\!]_k[\![\bot]\!]_k]_{k+1}]_{k+2})$. In the part in the dashed rectangle, $\mathcal{A}'$ simulates $\mathcal{A}$. However, instead of reading an input symbol $a \in \{\mathtt{0},\mathtt{1}\}$, $\mathcal{A}$ stores that symbol on the stack. In order not to interfere with the simulation of $\mathcal{A}$, this is done by copying the order-k stack used by $\mathcal{A}$ and storing a in the copy below. This is achieved as follows. For every edge $p \xrightarrow{a|\gamma|v} q$ with $v \in (I_k^\Gamma)^*$, $\mathcal{A}'$ instead has an edge

$$p \xrightarrow{\varepsilon|\gamma|\mathsf{push}_1\mathsf{rew}_a\mathsf{push}_{k+1}\mathsf{pop}_1 v} q$$

This pushes the input symbol a on the (topmost order-k) stack, makes a copy of the topmost order-k stack, removes the a from this fresh copy, and then excutes v. Edges $p \xrightarrow{\varepsilon|\gamma|v}$ (i.e. ones that read ε from the input) are kept.

When $\mathcal{A}'$ arrives in q_f, it has a stack $[[\![\bot']\!]_{k+1}[[\![\#]\!]_k s_1 \cdots s_m s]_{k+1}]_{k+2}$, where s is the order-k stack reached in the computation of $\mathcal{A}$, and $s_1, \ldots, s_m$ store the input word $w \in \Sigma^*$ read by $\mathcal{A}$, meaning $\mathsf{top}(s_1)\cdots\mathsf{top}(s_m) = w$. When moving to p, $\mathcal{A}'$ removes s so as to obtain $[[\![\bot']\!]_{k+1}[[\![\#]\!]_k s_1 \cdots s_m]_{k+1}]_{k+2}$ as a stack.

In p, $\mathcal{A}'$ reads the input word $\mathtt{0}^{\nu(w)}$ as follows. While in p, the stack always has the form

$$t = [[\![\bot']\!]_{k+1} t_1 \cdots t_\ell]_{k+2}, \tag{1}$$

where each t_i is an order-$(k+1)$ stack of the form $[[\![\#]\!]_k s_1 \cdots s_m]_{k+1}$ for some order-k stacks $s_1, \ldots, s_m \in \mathcal{S}_k^\Gamma$. To formulate an invariant that holds in state p, we define a function μ on the stacks as in (1). First, if $t_i = [[\![\#]\!]_k s_1 \cdots s_m]_{k+1}$, then let $\mu(t_i) = \nu(\mathsf{top}(s_1)\cdots\mathsf{top}(s_m))$. Next, let $\mu(t) = \mu(t_1) + \cdots + \mu(t_\ell)$. It is not hard to see that the loops on p preserve the following invariant: If $\mathtt{0}^r$ is the input word read from configuration (p, t) to (p, t'), then $\mu(t) = r + \mu(t')$. To see this, consider a one step transition $(p, t) \xrightarrow{\varepsilon|\mathtt{0}|\mathsf{pop}_{k+1}\mathsf{push}_{k+2}} (p, t')$. If

$$t = [[\![\bot']\!]_{k+1} t_1 \cdots t_\ell]_{k+2} = [[\![\bot']\!]_{k+1} t_1 \cdots t_{\ell-1}[[\![\#]\!]_k s_1 \cdots s_m]_{k+1}]_{k+2}$$

then

$$t' = [[\![\bot']\!]_{k+1} t_1 \cdots t_{\ell-1}[[\![\#]\!]_k s_1 \cdots s_{m-1}]_{k+1}[[\![\#]\!]_k s_1 \cdots s_{m-1}]_{k+1}]_{k+2}.$$

If $w = \mathsf{top}(s_1)\ldots\mathsf{top}(s_m)$ then $w = w'\mathtt{0}$ where $w' = \mathsf{top}(s_1)\cdots\mathsf{top}(s_{m-1})$ since we popped s_m off the stack. Moreover,

$$\mu(t') = \sum_{i=1}^{\ell-1} \mu(t_i) + 2\nu(w') = \sum_{i=1}^{\ell-1} \mu(t_i) + \nu(w) = \mu(t)$$

Similarly we see that if the transition taken is $\mathtt{0}|\mathtt{1}|\mathsf{pop}_{k+1}\mathsf{push}_{k+2}$ then we get $\mu(t) = \mu(t') + 1$. By induction on the length of the run, we get $\mu(t) = r + \mu(t')$ when $\mathtt{0}^r$ is read.

Now observe that when $\mathcal{A}'$ first arrives in p with stack t, then by construction we have $\ell = 1$ and $\mu(t) = \mu(t_1) = \nu(w)$. Moreover, when $\mathcal{A}'$ moves on to q_f' with a stack as in (1), then $\ell = 0$ and thus $\mu(t) = 0$. Thus, the invariant implies that if $\mathcal{A}'$ reads $\mathtt{0}^r$ while in p, then $r = \nu(w)$. This means, $\mathcal{A}'$ has read $\mathtt{1}\mathtt{0}^{\nu(w)}$ in total.

Finally, from a stack t as in (1), $\mathcal{A}'$ reaches q_f' in finitely many steps, please see the full version [40] for details. ◀

▶ **Lemma 5.5.** *The problem* $\mathsf{IE}(\mathcal{H}, \mathcal{I}(\mathsf{power}\mathcal{H}))$ *is undecidable.*

Proof. We reduce intersection emptiness for context-free languages, which is well-known to be undecidable [24], to $\mathsf{IE}(\mathcal{H},\mathcal{I}(\mathsf{power}\mathcal{H}))$. Let $K_1, K_2 \subseteq \{\mathtt{0},\mathtt{1}\}^*$ be context-free. Since $K_1 \cap K_2 \neq \emptyset$ if and only if $\mathtt{1}K_1 \cap \mathtt{1}K_2 \neq \emptyset$ and $\mathtt{1}K_i$ is context-free for $i = 0, 1$, we may assume that $K_1, K_2 \subseteq \mathtt{1}\{\mathtt{0},\mathtt{1}\}^*$. This implies $K_1 \cap K_2 \neq \emptyset$ if and only if $\nu(K_1) \cap \nu(K_2) \neq \emptyset$.

Let $P_2 = \mathbb{N} \setminus 2^{\mathbb{N}} \ \cup \ 2^{\nu(K_2)}$. Then $P_2 \subseteq \mathbb{N}$ is a power-$\mathcal{H}$ predicate, because $\mathcal{H}$ includes the context-free languages. Thus, the language $L_2 = \{\mathtt{10}^n \mid n \in P_2\}$ belongs to $\mathcal{I}(\mathsf{power}\mathcal{H})$ and

$$L_2 = \{\mathtt{10}^n \mid n \in \mathbb{N} \setminus 2^{\mathbb{N}}\} \cup \{\mathtt{10}^{2^{\nu(w)}} \mid w \in K_2\}.$$

Moreover, let $L_1 := \{\mathtt{10}^{2^{\nu(w)}} \mid w \in K_1\}$. Since $L_1 = \{\mathtt{10}^{\nu(\mathtt{10}^{\nu(w)})} \mid w \in K_1\}$ and K_1 is an order-1 pushdown language, applying Lemma 5.4 twice yields that L_1 is an order-5 pushdown language and thus belongs to $\mathcal{H}$. Now clearly $L_1 \cap L_2 \neq \emptyset$ if and only if $\nu(K_1) \cap \nu(K_2) \neq \emptyset$, which is equivalent to $K_1 \cap K_2 = \emptyset$. ◀

For showing decidability of regular separability, we use the following well-known fact (please see the full version [40] for a proof).

▶ **Lemma 5.6.** *Let $L = \bigcup_{i=1}^m L_i$ and $K = \bigcup_{i=1}^n K_i$. Then $K|L$ if and only if $L_i|K_j$ for all $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,n\}$.*

The last ingredient for our decision procedure is the following simple but powerful observation from [12] (for the convenience of the reader, a proof can be found in [40]).

▶ **Lemma 5.7.** *Let $K \subseteq \Gamma^*$, $L \subseteq \Sigma^*$ and $T \subseteq \Sigma^* \times \Gamma^*$ be a rational transduction. Then $L|TK$ if and only if $T^{-1}L|K$.*

The following now completes the proof of Theorem 5.2.

▶ **Lemma 5.8.** *The problem $\mathsf{RS}(\mathcal{H},\mathcal{I}(\mathsf{power}\mathcal{H}))$ is decidable.*

Proof. Suppose we are given $L_1 \subseteq \Sigma^*$ from $\mathcal{H}$ and $L_2 \subseteq \Sigma^*$ from $\mathcal{I}(\mathsf{power}\mathcal{H})$. Then we can write $L_2 = \bigcup_{i=1}^n T_i\mathtt{a}^{P_i}$, where for $1 \leq i \leq n$, $T_i \subseteq \Sigma^* \times \mathtt{a}^*$ is a rational transduction and $P_i \subseteq \mathbb{N}$ is a power-$\mathcal{H}$ predicate. Since $L_1|L_2$ if and only if $L_1|T_i\mathtt{a}^{P_i}$ for every i (Lemma 5.6), we may assume $L_2 = T\mathtt{a}^P$ for $T \subseteq \Sigma^* \times \mathtt{a}^*$ rational and $P \subseteq \mathbb{N}$ a power-$\mathcal{H}$ predicate. According to Lemma 5.7, $L_1|T\mathtt{a}^P$ if and only if $T^{-1}L_1|\mathtt{a}^P$. Since T^{-1} is also a rational transduction and $\mathcal{H}$ is a full trio, we may assume that L_1 is in $\mathcal{H}$ with $L_1 \subseteq \mathtt{a}^*$ and $L_2 = \mathtt{a}^P$.

By Lemma 4.5, we know that $L_1|\mathtt{a}^P$ if and only if L_1 is finite and disjoint from $\mathtt{a}^P$. We can decide this as follows. First, using Lemma 5.3 we check whether L_1 is finite. If it is not, then we know that $L_1|L_2$ is not the case.

If L_1 is finite, then we can compute a list of all words in L_1: We start with $F_0 = \emptyset$ and then successively compute finite sets $F_i \subseteq L_1$. For each $i \in \mathbb{N}$, we check whether $L_1 \subseteq F_i$, which is decidable because $L_1 \cap (\mathtt{a}^* \setminus F_i)$ is in $\mathcal{H}$ and emptiness is decidable for $\mathcal{H}$. If $L_1 \not\subseteq F_i$, then we enumerate words in $\mathtt{a}^*$ until we find $\mathtt{a}^m$ with $\mathtt{a}^m \in L_1$ (membership in L_1 is decidable) and $\mathtt{a}^m \notin F_i$. Then, we set $F_{i+1} = F_i \cup \{\mathtt{a}^m\}$. Since L_1 is finite, this procedure must terminate with $F_i = L_1$. Now we have $L_1|\mathtt{a}^P$ if and only if $F_i \cap \mathtt{a}^P = \emptyset$. The latter can be checked because $\mathsf{power}\mathcal{H}$ predicates are decidable. ◀

6 Conclusion

We have presented a language class $\mathcal{C}_1$ for which intersection emptiness is decidable but regular separability is undecidable in Section 4. Similarly, in Section 5 we constructed $\mathcal{C}_2, \mathcal{D}_2$ for which intersection emptiness is undecidable but regular separability is decidable. All three language classes enjoy pleasant language theoretic properties in that they are full trios and have a decidable emptiness problem.

Let us provide some intuition on why these examples work. The underlying observation is that intersection emptiness of two sets is insensitive to the shape of their members: If $f\colon X \to Y$ is any injective map and S disjoint from the image of f, then for $A, B \subseteq X$, we have $A \cap B = \emptyset$ if and only if $(f(A) \cup S) \cap f(B) = \emptyset$. Regular separability, on the other hand, is affected by such distortions: For example, if $K, L \subseteq \mathtt{1}\{\mathtt{0},\mathtt{1}\}^*$ are infinite, then $\mathtt{a}^{\mathbb{N}\setminus 2^{\mathbb{N}}} \cup \mathtt{a}^{2^{\nu(K)}}$ and $\mathtt{a}^{2^{\nu(L)}}$ are never regular-separable, even if K and L are. Hence, roughly speaking, the examples work by distorting languages (using encodings as numbers) so that intersection emptiness is preserved, but regular separability reflects infinity of the input languages. We apply this idea to language classes where intersection is decidable, but infinity is not (Theorem 4.1) or the other way around (Theorem 5.2). All this suggests that regular separability and intersection emptiness are fundamentally different problems.

Moreover, our results imply that any simple combinatorial decision problem that characterizes regular separability has to be incomparable with intersection emptiness. Consider for example the *infinite intersection problem* as a candidate. It asks whether two given languages have an infinite intersection. Note that if L and K are languages from $\mathcal{C}$ and $\mathcal{D}$, respectively, then $L \cap K \neq \emptyset$ if and only if $L\#^*$ and $K\#^*$ (where $\#$ is a symbol not present in L or K) have infinite intersection. Moreover, if $\mathcal{C}$ and $\mathcal{D}$ are full trios, then they effectively contain $L\#^*$ and $K\#^*$, respectively. This implies a counterexample with decidable regular separability and undecidable infinite intersection.

While the example from Section 4 is symmetric (meaning: the two language classes are the same) and natural, the example in Section 5 is admittedly somewhat contrived: While pseudo-$\mathcal{C}$ predicates rely on the common conversion of binary into unary representations, power-$\mathcal{C}$ predicates are a bit artificial. It would be interesting if there were a simpler symmetric example with decidable regular separability and undecidable intersection.

References

1 Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 313–321. IEEE, 1996.

2 Jorge Almeida. Some algorithmic problems for pseudovarieties. *Publ. Math. Debrecen*, 54(1):531–552, 1999.

3 Jean Berstel. *Transductions and context-free languages.* Springer-Verlag, 2013.

4 Achim Blumensath and Erich Gradel. Automatic structures. In *Proceedings of LICS 2000*, pages 51–62. IEEE, 2000.

5 Mikołaj Bojańczyk. It is Undecidable if Two Regular Tree Languages can be Separated by a Deterministic Tree-walking Automaton. *Fundam. Inform.*, 154(1-4):37–46, 2017.

6 Ahmed Bouajjani, Javier Esparza, and Tayssir Touili. A generic approach to the static analysis of concurrent programs with procedures. *International Journal of Foundations of Computer Science*, 14(04):551–582, 2003.

7 Christian Choffrut and Serge Grigorieff. Separability of rational relations in $A^* \times \mathbb{N}^m$ by recognizable relations is decidable. *Information processing letters*, 99(1):27–32, 2006.

8 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Regular Separability of Parikh Automata. In *Proceedings of ICALP 2017*, pages 117:1–117:13, 2017.

9 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Separability of Reachability Sets of Vector Addition Systems. In *Proceedings of STACS 2017*, pages 24:1–24:14, 2017.

10 Wojciech Czerwiński, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular Separability of Well-Structured Transition Systems. In *Proceedings of CONCUR 2018*, pages 35:1–35:18, 2018. doi:10.4230/LIPIcs.CONCUR.2018.35.

11 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A Characterization for Decidable Separability by Piecewise Testable Languages. *Discrete Mathematics & Theoretical Computer Science*, 19(4), 2017.

12 Wojciech Czerwiński and Georg Zetzsche. An Approach to Regular Separability in Vector Addition Systems, 2019. In preparation.

13 Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory.* Springer, Heidelberg, 1989.

14 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset Nets Between Decidability and Undecidability. In *Proceedings of ICALP 1998*, pages 103–115, 1998.

15 Herbert B Enderton. *A mathematical introduction to logic.* Elsevier, 2001.

16 Joost Engelfriet. Context-Free Grammars with Storage. *CoRR*, abs/1408.0683, 2014. arXiv:1408.0683.

17 Alain Finkel and Philippe Schnoebelen. Fundamental Structures in Well-Structured Infinite Transition Systems. In *Proceedings of LATIN 1998*, pages 102–118, 1998.

18 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Well-structured languages. *Acta Informatica*, 44(3-4):249–288, 2007.

19 Seymour Ginsburg and Sheila Greibach. Principal AFL. *Journal of Computer and System Sciences*, 4(4):308–338, 1970.

20 Samuel J.v. Gool and Benjamin Steinberg. Pointlike sets for varieties determined by groups. *Advances in Mathematics*, 348:18–50, 2019.

21 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding Piecewise Testable Separability for Regular Tree Languages. In *Proceedings of ICALP 2016*, pages 97:1–97:15, 2016.

22 Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311–324, 1978.

23 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and Downward Closures of Higher-order Pushdown Automata. In *POPL 2016*, pages 151–163, New York, NY, USA, 2016. ACM.

24 Juris Hartmanis. Context-free languages and Turing machine computations. In *Proceedings of Symposia in Applied Mathematics*, volume 19, pages 42–51, 1967.

25 Harry B. Hunt III. On the Decidability of Grammar Problems. *Journal of the ACM*, 29(2):429–447, 1982. doi:10.1145/322307.322317.

26 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *Logic and Computational Complexity*, volume 960 of *LNCS*, pages 367–392, Berlin, Heidelberg, 1995. Springer.

27 Felix Klaedtke and Harald Rueß. Monadic Second-Order Logics with Cardinalities. In *Proceedings of ICALP 2003*, volume 2719 of *LNCS*, pages 681–696, Springer, Heidelberg, 2003. Springer.

28 Eryk Kopczyński. Invisible Pushdown Languages. In *Proceedings of LICS 2016*, pages 867–872, New York, NY, USA, 2016. ACM.

29 Sławomir Lasota and Wojciech Czerwiński. Regular Separability of One Counter Automata. *Logical Methods in Computer Science*, 15, 2019. Extended version of LICS 2017 paper.

30 Richard Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1-3):337–354, 2003.

31 Rohit J Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

32 Thomas Place. Separating Regular Languages with Two Quantifiers Alternations. In *Proceedings of LICS 2015*, pages 202–213, 2015.

33 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating Regular Languages by Piecewise Testable and Unambiguous Languages. In *Proceedings of MFCS 2013*, pages 729–740, 2013.

34 Thomas Place and Marc Zeitoun. Separation and the Successor Relation. In *Proceedings of STACS 2015*, pages 662–675, 2015.

35 Thomas Place and Marc Zeitoun. Separating Regular Languages with First-Order Logic. *Logical Methods in Computer Science*, 12(1), 2016.

36 Thomas Place and Marc Zeitoun. Concatenation Hierarchies: New Bottle, Old Wine. In *Proceedings of CSR 2017*, pages 25–37, 2017.

37 Thomas Place and Marc Zeitoun. Separation for dot-depth two. In *Proceedings of LICS 2017*, pages 1–12, 2017.

38 Thomas Place and Marc Zeitoun. The Covering Problem. *Logical Methods in Computer Science*, 14(3), 2018.

39 Thomas G. Szymanski and John H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal on Computing*, 5(2), 1976.

40 Ramanathan S Thinniyam and Georg Zetzsche. Regular Separability and Intersection Emptiness are Independent Problems. *arXiv preprint*, 2019. `arXiv:1908.04038`.

41 Georg Zetzsche. An Approach to Computing Downward Closures. In *Proceedings of ICALP 2015*, pages 440–451, 2015.

42 Georg Zetzsche. Separability by piecewise testable languages and downward closures beyond subwords. In *Proceedings of LICS 2018*, pages 929–938, 2018.